Universidad Carlos III de Madrid
Departamento de ingeniería de sistemas y automática

Grado en Electrónica Industrial y Automática
2014-2015

Trabajo Fin de Grado

"Implementación de algoritmo de scan-matching basado en PSO"

Autor:
Carlos Gallo Merino

Tutor:
Dr. Fernando Martín Monar

Leganés, 7 de Julio de 2015

Agradecimientos

Me gustaría mostrar mi agradecimiento a todas las personas que han estado cerca de mí durante esta etapa de mi vida, esos que me han ayudado y apoyado durante este camino, en especial a grandes compañeros como Álvaro, Rodrigo y Jaime.
Agradecer también al Dr. Fernando Martin que no solo me ha guiado en el presente proyecto si no que me ha enseñado todo lo que se acerca del control de sistemas. Me gustaría agradecer también a la Universidad Carlos III y a su profesorado todo el conocimiento que me han aportado durante estos años.
Por último agradecer todo lo que tengo y lo que soy a lo que más quiero, mi familia.

Resumen

Uno de los campos en los que se está centrando la robótica los últimos años es el desarrollo de nuevos algoritmos que permitan localizar el robot en un entorno con el objetivo de mejorar su autonomía.

Este problema ha llevado al desarrollo de algoritmos de SLAM (Simultaneous Localization and Mapping) que buscan localizar al robot mientras se construye un mapa de su entorno y a la vez se explora el mismo usando este mapa. Un aspecto relativo a esta tarea en particular es el problema de scan matching, que es el objeto principal de este trabajo. El problema de scan matching cosiste en estimar la relación métrica (en posición y orientación) entre pares de escaneos, utilizando la información proporcionada por estos. Este es un problema muy común en robots móviles debido a que a menudo la información nos es dada por sensores extereoceptivos con información de profundidad.

En este caso ha desarrollado una solución al Scan Matching o correlación de escaneos basada en algoritmo del PSO (Particle Swarm Optimization). Se utilizará además, las propiedades del color para optimizar el proceso. La utilización del color permitirá un ahorro en coste computacional.
Este nuevo algoritmo se ha probado en un entorno real tridimensional con escaneos proporcionados por el sensor Kinect y se ha conseguido resolver el problema de scan matching.

.

Abstract

One of the aspects on which robotics has been focused in the last years is the development of new algorithms that allow the localization of a mobile robot in order to improve its autonomy.

This problem has caused the development of new SLAM (Simultaneous Localization and Mapping) algorithms that try to track the robot's pose while building a map of the environment and exploring it at the same time using this map. A particular aspect of this task is the scan matching problem, which is the main topic of this work. The scan matching problem consists of estimating the metric relation (position and orientation) between a pair of scans using the information provided by these scans. This is a very common problem in mobile robotics because the information about the environment that is going to be represented is often given by an extereoceptive sensor with depth information.

A solution for the Scan Matching problem based on PSO (Particle Swarm Optimization) has been implemented. It also uses colour features in order to improve the algorithm performance. Using colour proprieties the computational cost will be reduced.
This new algorithm has been tested in a 3D real environment with scans given by the Kinect sensor and it can solve the scan matching problem.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Contexto

En la actualidad cuando hablamos de robot lo que primero se nos viene a la cabeza es un robot con morfología humanoide que es capaz de desplazarse, manipular objetos, comunicarse y en definitiva interactuar con todo lo que hay en el entorno que le rodea.
Probablemente esta imagen preconcebida viene de robots como el *ASIMO* desarrollado por Honda o el *Atlas* de Boston Dynamics (compañía que pertenece a Google), sin embargo existe un gran número de tipos robots, y su cantidad ha ido creciendo durante las últimas décadas gracias al desarrollo de las nuevas tecnologías. De la misma forma que incrementa el número de robots lo hacen las definiciones de "robot" que cada vez incluyen mas capacidades y competencias. Y es que si comparamos algunos de los primeros automatismos como el *Teatro automático* de Herón de Alejandría (62 a. C) con los últimos trabajos de la empresa nombrada anteriormente Boston Dynamics, vemos como las ideas "ficticias" de Karel Kapek (1890-1938) o Isaac Asimov (1920-1992) acerca de los robots no están muy alejadas de la realidad.

El crecimiento de la robótica desde las últimas décadas hace difícil prever la situación que esta adquirirá los próximos años, sin embargo alguno de los últimos estudios, como el presentado por "Business Insider" [1] vaticina que la cifra de robots de consumo demandados para 2019 rondaría los 6 millones con un valor en el mercado de unos 1.5 billones de euros. El estudio nos presenta las 3 fuerzas que impulsarán este crecimiento:

- El desarrollo de tecnologías de asistencia como el *Siri* de Apple o *Google Now*.

- El crecimiento de Internet, de terminales móviles y aplicaciones, que permitirían el control de los mismos.

- Los avances en sistemas de navegación y de inteligencia artificial que consiguen una mayor autonomía alejándose cada vez más de la asistencia y supervisión humana.

Figura 1.1: a la izquierda el robot *ASIMO* de Honda (2000) y a la derecha el robot *Atlas* desarrollado por Boston Dynamics (2013)

Dentro de ésta última característica encontramos la inteligencia artificial, foco en el que se centra el presente proyecto. Una buena definición de en qué consiste la I.A. la dan Ritch y Knight [2] diciendo que es "El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor". La I.A. comprende numerosas competencias, una de ellas es la "visión por computador" o "visión artificial".

Podemos definir la *visión artificial* como una rama de la *inteligencia artificial* que mediante el uso de una serie de técnicas permite la obtención, procesamiento y análisis de cualquier tipo de información obtenida mediante imágenes digitales.

En el pasado, enfrentarse a estos retos técnicos solo era posible apoyándose en sensores de gran calidad y precisión, lo que implicaba que la totalidad de lo que se desarrollaba se limitaba al ámbito de la investigación ya que el precio del material y hardware necesario hacía inviable su comercialización al público. Sin embargo, gracias al crecimiento en las últimas décadas de áreas como la visión artificial y el resto de nuevas tecnologías se ha podido

renunciar al uso de sensores y materiales tan sumamente costosos sustituyéndose por otros más baratos y nuevas técnicas de programación e interpretación de los datos recogidos. Por lo que el problema está pasando de ser económico a técnico.

Un ejemplo a lo planteado es el desarrollo por parte de algunas marcas de automóviles por crear un vehículo autónomo a un precio de mercado viable. Ya se ha conseguido desarrollar un vehículo autónomo, véase el coche de Google, sin embargo el precio de cada unidad rondaría los 150 mil euros lo que lo hace económicamente inviable para su comercialización.

Es por esto que las marcas de automóviles se han puesto manos a la obra para conseguir prestaciones lo más cercanas posibles mediante sensores más económicos, como la disposición de cámaras, e invertir en el desarrollo de algoritmos de visión artificial que sepan interpretar y reducir los errores proporcionados por los mismos.
Este es solo uno de los ejemplos de aplicación de una de las técnicas robóticas y de automatización que más proyección tienen.

Uno de los retos que plantea la robótica es, como hemos dicho antes, buscar la interacción con el entorno lo más fluida y precisa posible. Para lograr esto, una buena localización del robot es sin duda algo esencial. Imaginemos, por ejemplo, que pretendemos diseñar un robot para rescate de personas en catástrofes, el robot debería ser capaz de moverse en entornos difíciles y de localizar a las víctimas, para hacer todo esto debería saber en qué punto se encuentra en todo momento. Este problema planteado en exteriores puede afrontarse fácilmente con el uso de un GPS, sin embargo, es en interiores donde la imposibilidad del uso del mismo (debido a la perdida de precisión) hace necesaria la utilización de visión artificial.

Una de las soluciones en visión artificial que se plantea para resolver este problema es la construcción de un mapa que sirva de apoyo para localizar tanto lo que rodea al robot como el punto en el que este se encuentra en todo momento.

1.2. Objetivo

Un robot completamente autónomo debe ser capaz de: obtener información del entorno, trabajar durante un tiempo prolongado sin necesidad de la intervención humana, ser capaz de

moverse a lo largo del entorno sin asistencia humana, evitar situaciones peligrosas y procurar su propia supervivencia sin romper las reglas anteriores.
Como se ha visto antes, conocer la localización del robot y el mapa en que se sitúa en todo momento será un pilar básico en la construcción de un robot autónomo. A la resolución de este problema se conoce como *maping* o mapeado.

El mapeado esta cercanamente relacionado con el problema de SLAM (Simultaneous Localization and Mapping) introducido originalmente por Leonard y Durant-White [3] basándose en el trabajo anterior de Smith *et al.* [4]. El problema de SLAM en robots móviles consiste en la construcción de un mapa de un entorno desconocido mientras a la vez se explora el mismo, usando este mapa.
El trabajo que se desarrolla en este proyecto esta focalizado en el scan matching (o correlación de escaneos) que es un problema particular del mapeado. El scan matching se puede definir como la estimación de la relación métrica en posición y orientación entre pares de escaneos, usando la información aportada por esos escaneos.
Este es un problema comúnmente abordado en robótica ya que la información que se busca representar, es, a menudo, proporcionada por sensores exteroceptivos con información compleja.

El objetivo de este proyecto es presentarse como una herramienta más de ayuda al mapeado para localización robótica y su comparación con los algoritmos ya existentes para ver las ventajas/desventajas que presenta frente a ellos.

El punto de partida será el trabajo desarrollado por Martín *et al.* [5], en el se analiza la influencia del color al utilizarse como elemento característico a la hora de resolver el problema de scan matching e implementándose mediante el algoritmo de DE (Differential Evolution). En este caso se procederá a implementar mediante otro método evolutivo, el algoritmo del PSO (Particle Swarm Optimization) y analizar su resultado.

Para tomar el color como característica en la distinción de puntos se utilizara la cámara "Kinect" desarrollada por Microsoft que llevara integrada el robot.
Para llevar a cabo esta correlación nos ayudaremos de las propiedades de los colores del entorno que es capaz de captar la cámara y que nos servirá como apoyo para ajustar en mayor medida los escaneos.

El proyecto se presentara de la siguiente forma:

- En el *capítulo 2* se presentará el **estado del arte** donde se expondrán los conceptos teóricos básicos respecto del problema de scan matching, las diferentes formas de correlación de escaneos según el tipo de datos a tratar, así como los métodos más utilizados para su resolución.
- En el *capítulo 3* se presentara el método **PSO** (optimización por nube de partículas) partiendo de la idea originaria y su pseudo-código, se presentarán también sus evoluciones y sus posibles configuraciones.
- En el *capítulo 4* se expondrá el algoritmo desarrollado para solucionar el problema de **scan matching en 6D mediante PSO.**
- En el *capítulo 5* se presentarán los **experimentos** realizados y el **resultado** obtenido.
- En el capítulo 6 se expondrán las **conclusiones** sobre los resultados obtenidos y los **posibles trabajos futuros.**

Capítulo 2

Estado del arte

2.1. Scan matching

Una de las técnicas más usadas en la actualidad para construir mapas consistentes es el scan matching. Este método se hace necesario debido al gran error producido por las técnicas basadas únicamente en el mapeado por odometría.

La odometría mide el desplazamiento del robot en función de la rotación de sus ruedas. Ésta técnica se ha venido usando durante años para estimar la posición de los robots debido a su bajo precio y su facilidad de implementación. Sin embargo, a pesar de que a corto plazo puede presentar un error asumible, al ser una técnica que maneja información del movimiento de forma incremental, a largo plazo el error se hace inadmisible impidiendo una buena localización del robot.

Utilizando técnicas odométricas se presentan algunos problemas como que el propio movimiento presenta ruido: el desplazamiento está condicionado por el terreno e implica deslizamiento de las ruedas que provoca errores en las medidas, además, las ruedas nunca son exactamente iguales ni están perfectamente equilibradas, por otro lado, los sensores no son perfectos por lo que, aunque redujéramos al mínimo los posibles errores en la toma de medidas, según el robot vaya en movimiento, estos errores irán incrementándose de forma que el mapa creado será poco fiel para realizar según qué tareas. La utilización del scan matching servirá por lo tanto, para reducir el error producido por métodos odométricos.

El método de scan matching se basa en hacer un escaneo del entorno y compararlo con un modelo o un escaneo anterior de forma que mediante una trasformación basada en rotación y traslación estos escaneos queden lo más alineado posible, reduciendo al máximo el error en la localización del robot y haciendo posible la construcción de un mapa fiel a la realidad.

El scan matching o registro puede clasificarse según distintos tipos: según las dimensiones del mapa, según el tipo de escaneos a hacer corresponder y según el método seguido para hacer corresponder los mismos.

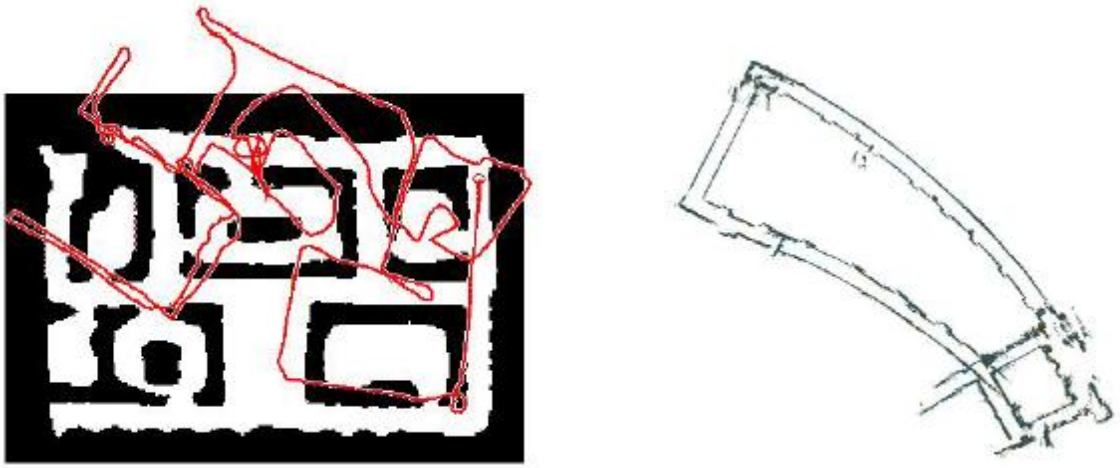

Figura 2.1: En las figuras tomadas [6] se pueden apreciar los errores producidos por el mapeado basado en odometría. En la figura de la izquierda se ve como errores reducidos en un principio producen grandes errores a lo largo del tiempo. En la figura de la derecha vemos el error que aparece en entornos cíclicos, para los cuales, se hace necesario que se establezcan correspondencias entre posiciones pasadas y presentes y así lograr un correcto mapeado.

2.1.1. Según las dimensiones del mapa

Los mapas podrán ser de 2D o de 3D. Los mapas de 3D implicaran mayor coste computacional debido a que manejaran mayor cantidad de información. El presente proyecto está basado en un mapa de 3D.

2.1.2. Según tipo de escaneos a hacer corresponder

El scan matching también se podrá realizar mediante métodos locales o globales, los primeros buscan la correspondencia entre escaneos singulares mientras que los globales lo harán entre el escaneo en el instante y el modelo global.

- Métodos locales: los métodos locales o secuenciales se basan en hacer corresponder dos escaneos sucesivos tomando un primer escaneo como escaneo de referencia con su posición de referencia y un segundo como escaneo nuevo [7]. El proceso seguido es llevar el escaneo previo en la posición nueva y entonces buscar la correspondencia entre puntos del escaneo previo y el nuevo. Este proceso presenta el problema de producir un cierto error que puede ir incrementando a lo largo del proceso del mapeado del entorno.
- Métodos globales: los métodos globales buscan la correspondencia entre el escaneo en el instante y un modelo global [8]. Son modelos fiables que presentan pocos problemas en las zonas curvas y frente a oclusión pero los modelos utilizados no pueden presentar errores ya que de tenerlos se trasladaran al proceso de mapeado.

2.1.3. Según el método de alineación de escaneos

La forma en la que los escaneos van a analizarse y hacerse corresponder también determina distintos caminos hacia un correcto scan matching. Existen métodos basados en buscar la correspondencia de elementos característicos, en este caso estos deben ser correctamente elegidos, ya que de ello dependerá la exactitud con la que se alinearan los escaneos. Como elementos característicos se suelen tomar esquinas o segmentos [9].

Otra forma de buscar la correspondencia es mediante la correlación entre puntos, esta forma es la más utilizada, y no necesita de una búsqueda de elementos característicos, algo que en algunos entornos se torna difícil. Por último existen técnicas que toman tanto algunos elementos característicos como puntos, por lo que sería un método hibrido de los dos anteriores. Debido a que el presente proyecto está basado en la correspondencia de puntos se va a presentar su variante más utilizada y que será seguida en éste proyecto: el método ICP; también se presentara otro método seguido por muchos grupos de investigación: el método NDT.

2.2. El método ICP (Iterative Closest Point)

Este método [10] trata de registrar conjuntos de puntos en un sistema de coordenadas común. Para ello se parte de dos conjuntos o nubes de puntos, el primero llamado M (el modelo) y el segundo llamado D (los datos) que corresponde a una escaneo en el instante. Se

tratara de buscar la transformación que incluye rotación R y traslación t que minimiza la siguiente función:

$$E(R,t) = \sum_{i=1}^{|M|} \sum_{j=1}^{|D|} w_{i,j} \left\| m_i - \left(Rd_j + t\right) \right\|^2 \qquad (2.1)$$

En la fórmula $w_{i,j}$ toma el valor de 1 si el punto i del conjunto M describe el mismo punto en el espacio que el j perteneciente a D. En otro caso el valor de $w_{i,j}$ será 0.

Siguiendo este método se deberán calcular en primer lugar los puntos más cercanos de correspondencia entre escaneos y en segundo lugar la transformación (R,t) que minimiza la función de coste $E(R,t)$ para las correspondencias entre dichos puntos.

El algoritmo del ICP quedaría de la siguiente forma [11]:

- Entradas: el algoritmo recibirá 2 imágenes en 3D conteniendo *m* y *n* puntos en 3D.
- Salidas: tras la iteración se logrará el movimiento óptimo entre las 2 imágenes.
- Procedimiento:
  1. Inicialización: se inicializaran las 2 nubes de puntos: modelo y datos.
  2. Iteración:
     a. Se computan los puntos más cercanos de las 2 nubes
     b. Se computa el registro o "matching" de esos puntos, es decir, se calcula la transformación (R,t) que minimiza la función de coste.
     c. Se actualiza el conjunto de datos aplicando la transformación
     d. La iteración finalizará cuando el error cuadrático de aplicar esa transformación sea menor que el máximo establecido.

De esta manera se van calculando de forma iterativa si los puntos del modelo y datos coinciden y de ser así se calculan la rotación y traslación que minimizan la función anterior. Se deberá asumir que la última iteración es la correcta.

Las principales ventajas que nos aporta este método [12] son:

- Da resultado en entornos con 6 grados de libertad
- No requiere preprocesamiento de datos, suavizado ni filtros especiales.
- No se necesita hacer una extracción de características previa.
- El coste computacional no es demasiado grande pudiéndose usar en tiempo real. Además este coste es previsible basándose en la complejidad de las formas (para alineamiento global) y en las oclusiones (alineamiento local).

- Es muy versátil ya que es fácil de implementar junto con otros algoritmos que produzcan un sistema más robusto para cada aplicación.

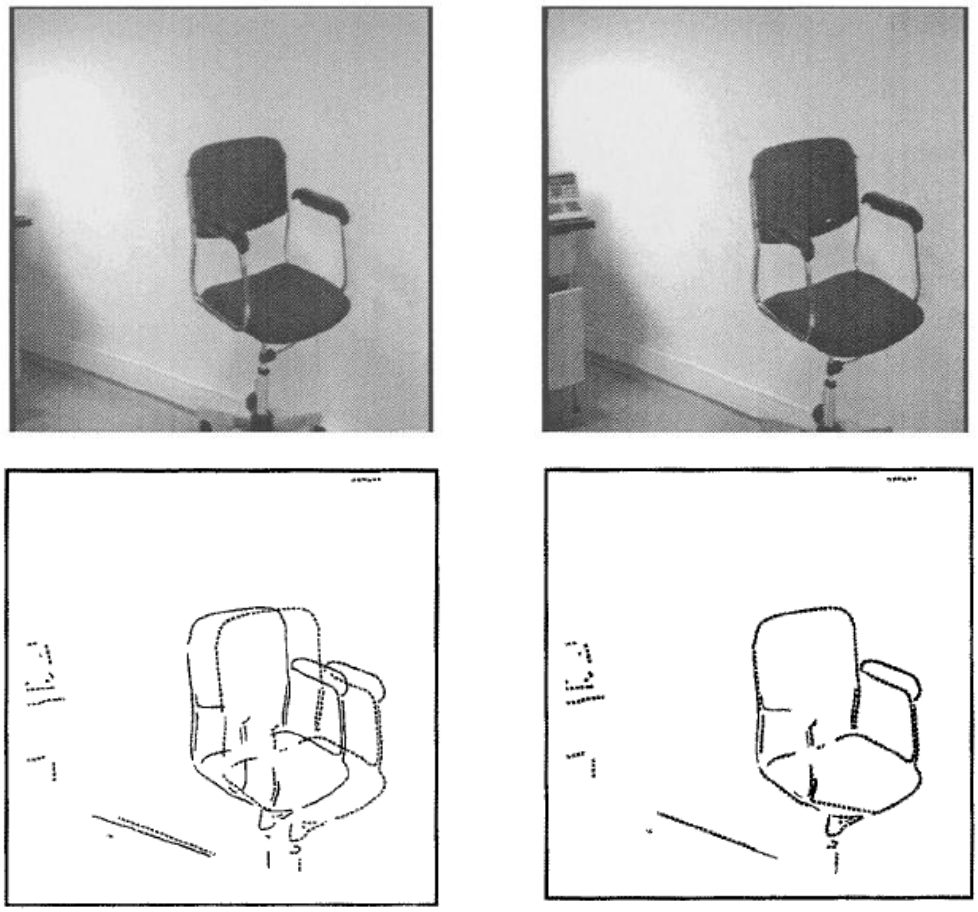

Figura 2.2: arriba: 2 imágenes tomadas desde distinta posición; abajo: superposición de las 2 imágenes anteriores antes y después de aplicar la transformación [11].

Las principales ventajas que nos aporta este método [12] son:

- Da resultado en entornos con 6 grados de libertad
- No requiere preprocesamiento de datos, suavizado ni filtros especiales.
- No se necesita hacer una extracción de características previa.
- El coste computacional no es demasiado grande pudiéndose usar en tiempo real. Además este coste es previsible basándose en la complejidad de las formas (para alineamiento global) y en las oclusiones (alineamiento local).
- Es muy versátil ya que es fácil de implementar junto con otros algoritmos que produzcan un sistema más robusto para cada aplicación.

Algunas variantes al método original y aplicadas en 3D son las que recoge Rusinkiewicz et al [13]. Estas variantes abarcan los grandes puntos del algoritmo ICP. Las clasifica según: la selección de los puntos de las nubes, la forma de emparejar los puntos de las distintas nubes, la manera de ponderar los emparejamientos de puntos, la forma de rechazo de los malos emparejamientos, la asignación del error en la medida basado en los pares de puntos y en la forma de minimizar el error en esta medida.

2.3. El método NDT (Normal Distributions Transform)

Otro método destacable es el propuesto por Biber et al [14], un método fiable y significativamente distinto al ICP. Éste método mide la probabilidad de la existencia de un punto en una posición determinada según una serie de distribuciones normales. De manera similar al método de celdas de ocupación, este método busca subdividir el plano. La diferencia con el método de celdas de ocupación es que éste representa simplemente la probabilidad de estar una celda ocupada o no y el NDT representa la probabilidad de medir una muestra para cada posición dentro de la celda, lo que aumenta la precisión en el análisis de datos.

En entornos en 3D el espacio se divide en *octrees* (árboles octales), en los que cada nodo cúbico está dividido en 8 octantes, para los cuales se hallará su media y matriz de covarianza de acuerdo con los puntos en su interior. Los cubos representan una distribución normal. El método introducirá los cubos del segundo escaneo dentro del previo, después se evalúa la probabilidad de acuerdo con la comparación de la distribución normal de los cubos donde se han situado. El resultado será la suma de las probabilidades. El proceso que seguirá será introducir rotaciones y traslaciones a los puntos hasta que el resultado se minimice siguiendo el algoritmo de Newton. Se utilizara el resultado del NDT como función de ajuste [5]:

$$e = -\sum_{j=1}^{N} exp\left[-\frac{(d_j - q_i)\Sigma^{-1}(d_j - q_i)^T}{2}\right] \qquad (2.2)$$

Donde N es el número total de puntos del segundo escaneo, j es cada uno de esos puntos, d son las coordenadas de los puntos, q es la media y Σ^{-1} es la matriz de covarianza de los puntos del escaneo previo situados en su cubo que contienen las d del segundo escaneo.

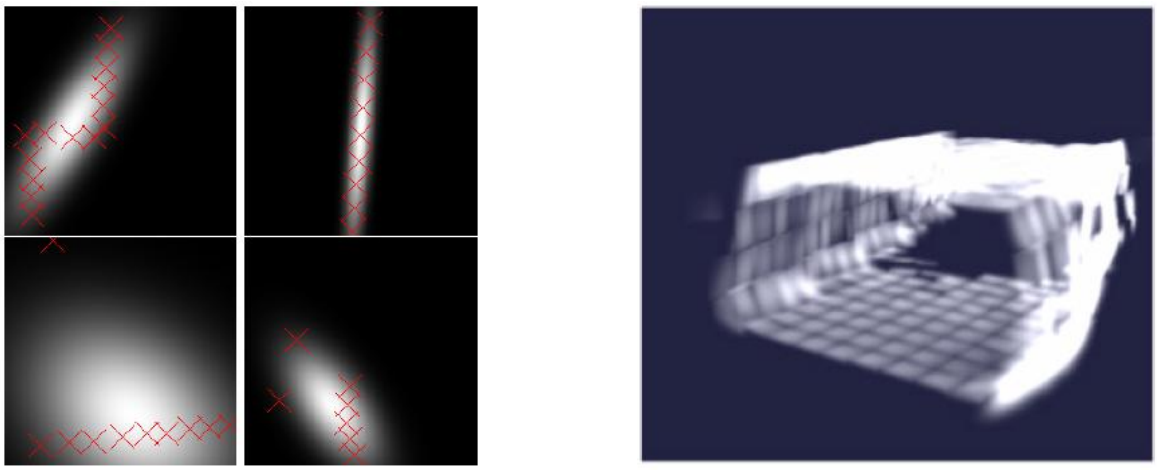

Figura 2.3: A la izquierda: NDT de Biber et al [14], en la figura se puede ver un ejemplo de la transformación de celdas singulares. Se muestran puntos en 2D y la densidad de población probabilística de su existencia. Las zonas luminosas representan la densidad de probabilidad alta. A la derecha: se puede ver una representación del método 3D-NDT [15] de un trozo de túnel de una mina en la que las partes luminosas representan igualmente densidad probabilística alta.

Una de las principales ventajas que presenta Magnuson *et al*. [15] en su aplicación del NDT para mapas en 3D en comparación con el ICP es que almacenar los datos escaneados mediante distribuciones normales combinadas es mucho más eficiente que almacenar las nubes de puntos completas. Esto en entornos pequeños no afecta demasiado, pero en entornos grandes y dinámicos, almacenar nubes de puntos a lo largo del tiempo puede complicar el proceso.

Capítulo 3

Fundamentos del PSO

3.1. Modelo del PSO

En este capítulo se van a presentar las ideas principales de este método de optimización, así como su algoritmo. Se expondrán también algunas de las evoluciones y modificaciones principales que mejoran el algoritmo original.

3.1.1. Introducción

Si nos remitimos al significado literal del método "PSO": "Particle Swarm Optimization" vemos que su traducción es "Optimización de nube de partículas". El término fue introducido por J.Kennedy, un psicólogo social y R.Eberhart, un ingeniero eléctrico, en 1995 [16]. Su trabajo se baso en aplicar la inteligencia de un comportamiento social en contraposición de las habilidades cognitivas individuales para la optimizar la resolución de un problema.
El origen de la idea de J.Kennedy y R.Eberhart sobre comenzar a trabajar con comportamientos sociales fue tomado de Heppner y Grenander [17], en el cual se hablaba sobre el comportamiento de los pájaros y su movimiento en bandadas y cómo el comportamiento de cada pájaro influía en la población y el de la población en cada pájaro.

El comportamiento en el que se basa el PSO se puede asemejar al comportamiento natural de la búsqueda de alimento de abejas, pájaros u hormigas. Cada elemento individual, cada abeja, buscará el lugar con mayor cantidad o calidad de alimento y evaluará con el grupo si el sitio

encontrado por ella es mejor o peor que los encontrados hasta el momento, y dependiendo de ello, el conjunto de abejas se acercara o no a la zona que la abeja había
Encontrado. Este comportamiento no tiene sentido con una única abeja o partícula, pero en un conjunto grande se es capaz de optimizar el trabajo y encontrar el sitio más apropiado rápidamente.

3.1.2. De inteligencia natural a inteligencia artificial

El comportamiento animal se ha estudiado ampliamente a lo largo de la historia, buscando comprender por qué actúan de la forma en la que lo hacen, cómo actúan, en que se basan para actuar de una manera u otra, cómo se organizan, etc. Es esta última cuestión la que se aborda en este proyecto, la forma de organización desarrollada por su instinto para optimizar su comportamiento.

A simple vista podemos ver dos formas muy comunes de comportamiento en los animales. Una primera es la que está regida por un líder, llamado normalmente macho o hembra alpha, que se encarga de tomar las decisiones que afectan al conjunto y que el resto del grupo acata. Este comportamiento lo podemos ver en manadas de leones, elefantes, monos, etc. Sin embargo el comportamiento que a nosotros nos interesa es el seguido por los grupos que no tienen un líder. En estos grupos cada individuo no sabe cuál es el comportamiento del resto de integrantes, ni sabe cuál es la mejor opción a tomar, y tampoco son conocedores de todo su entorno. En lugar de esto, ellos saben una pequeña parte de todo lo anteriormente nombrado, lo que únicamente les afecta a ellos, sin embargo, interaccionando "localmente" con el resto de individuos, esta información ira afectando progresivamente a todo el grupo creándose un comportamiento "global". De este modo funcionan las bandadas de pájaros, los bancos de peces o los insectos, y en su forma de actuar se basa el PSO.

El PSO tiene su origen más primitivo en el trabajo realizado por Reynolds [18]. Desarrolló un sistema simple basado en la "coreografía" seguida en el movimiento de las bandadas de pájaros. Cada partícula determinaba los individuos más cercanos y copiaba su velocidad resultando un movimiento sincronizado de la bandada. El problema que aparecía es que en su simulación las velocidades se corregían muy rápido y los pájaros no cambiaban su dirección. La solución fue introducir componentes aleatorios que alterasen el movimiento de los pájaros.

Una evolución hacia este concepto fue la introducción del término “rooster” [17], que incluía la idea de mejor posición y espacio anteriores. Estas son, la mejor posición encontrada por cada individuo desde el inicio de la simulación y la mejor encontrada por todo el grupo. Se vio que en unas pocas iteraciones la bandada de pájaros o nube rondaba el lugar objetivo. Esto se conseguía incluso sin ajustar sus velocidades.

El modelo que resultó fue el llamado “Particle Swarm Optimization” u optimización por nube de partículas. El término partículas viene de tomar los individuos como componentes sin masa ni volumen, únicamente como elementos con velocidad y aceleración.
El sistema era capaz entonces de adaptarse a entornos cambiantes, actualizándose los valores de mejor posición individual y mejor posición en el espacio de partículas, lo que permitía afrontar problemas de optimización.

3.1.3. Método del PSO

Desde la introducción del PSO este ha sufrido numerosas modificaciones y mejoras, que lo han llevado a ser utilizado para numerosos problemas como una herramienta muy potente. En un primer momento, en los primeros años tras su concepción, los investigadores se mostraron en cierto modo reacios a su utilización, acusándole incluso de no ser un algoritmo evolutivo, sin embargo, los últimos años y tras numerosas evoluciones, se está siendo consciente del gran poder que tiene y de su versatilidad respecto a la gran cantidad de problemas en los que se puede aplicar.

El PSO comprende una nube de partículas representando cada partícula una solución potencial al problema. Como se ha comentado antes, cada partícula representara a un individuo y al conjunto de la población de partículas se le llamara “swarm” o nube.

El objetivo que persigue el PSO será lanzar un número determinado de partículas al espacio de búsqueda. Una vez en el espacio cada una de ellas evaluara la función objetivo en su localización actual, determinando si su posición actual es mejor o peor que las encontradas hasta el momento. Cada partícula determinará su próximo movimiento por el espacio de búsqueda combinando elementos de sus localizaciones anteriores más óptimas con las más óptimas del resto de las partículas de la nube y con algunos elementos aleatorios.

A partir de este comportamiento, a lo largo de una serie de iteraciones las partículas encontraran el lugar más óptimo.

Cada partícula contará con un número *D* de vectores dimensionales donde *D* es la dimensión del espacio de búsqueda. Estos serán la posición actual, llamada $\overrightarrow{x_i}$, la mejor posición anterior $\overrightarrow{p_i}$ y la velocidad $\overrightarrow{v_i}$.
La posición $\overrightarrow{x_i}$ constara de las coordenadas según las dimensiones del problema. En cada iteración la nueva posición que se toma será evaluada, en el caso de que la posición fuera más favorable que la anterior esta será guardada en el segundo vector $\overrightarrow{p_i}$. Las coordenadas de la mejor posición encontrada hasta el momento se guardara en una variable llamada $pbest_i$ y servirá como comparación en iteraciones posteriores.
El objetivo es ir actualizando tanto $\overrightarrow{p_i}$ como $pbest_i$ con mejores posiciones hasta llegar a la solución final mas ajustada posible, para ello se introduce una variable con un componente aleatorio, la velocidad de las partículas $\overrightarrow{v_i}$, que producirá nuevas posiciones $\overrightarrow{x_i}$ distintas en cada iteración.

Como antes hemos comentado el poder del algoritmo reside en el poder de la nube completa de partículas y no el que tienen las partículas individualmente, la solución solo se encontrará si las partículas actúan como un conjunto, como una red social de transporte de información.

El comportamiento que sigue esta red social está basado en la comunicación entre pares partículas que se encuentran en entornos cercanos valorando sus posiciones y transmitiéndose esta información de partícula en partícula hasta que la tenga toda la nube y el poder del conjunto tome decisiones hacia qué posición tomar. Es por esto que dentro de las mejores posiciones tomadas por cada partícula $\overrightarrow{p_i}$, habrá una posición que será la mejor del entorno de entre todas las mejores individuales llamada $\overrightarrow{p_g}$.
Para valorar nuevas posiciones más favorables la velocidad de cada partícula se ajusta de forma iterativa para que oscile en los entornos de $\overrightarrow{p_i}$ y $\overrightarrow{p_g}$.

Cada individuo se tomará como un sistema dinámico, ya que, cada partícula estará en continuo movimiento, en continuo cambio. La dirección de su movimiento será función de la posición actual y de la velocidad, de la localización de la mejor posición previa de la partícula individualmente y de la mejor de entre todas las partículas:

$$\overrightarrow{x_i} \leftarrow \overrightarrow{x_i} + \overrightarrow{v_i} \tag{3.1}$$

Por otro lado como se ha comentado arriba la velocidad es función de la diferencia entre el mejor individual anterior y la posición actual, y la diferencia entre el mejor de entre todos los individuos y la posición actual.

$$\overrightarrow{v_i} \leftarrow \overrightarrow{v_i} + \vec{U}(0, \emptyset_1) * (\overrightarrow{p_i} - \overrightarrow{x_i}) + \vec{U}(0, \emptyset_2) * (\overrightarrow{p_g} - \overrightarrow{x_i}) \tag{3.2}$$

O de una forma equivalente:

$$\overrightarrow{v_i} \leftarrow \overrightarrow{v_i} + \varphi_1 * (\overrightarrow{p_i} - \overrightarrow{x_i}) + \varphi_2 * (\overrightarrow{p_g} - \overrightarrow{x_i}) \tag{3.3}$$

Donde las variables φ son números aleatorios definidos por un valor máximo. El efecto que esto tiene es que la partícula se mueve de manera desigual alrededor del punto definido como una media de las dos "mejores posiciones".

$$\frac{\varphi_1 \overrightarrow{p_i} + \varphi_2 \overrightarrow{p_g}}{\varphi_1 + \varphi_2} \tag{3.4}$$

Debido a esta aleatoriedad de los números la localización exacta de este punto cambia en cada iteración.

Por razones que luego se verán, la tendencia con este método es que las exploraciones sean cada vez más separadas, mas globales en el espacio, con cambios bruscos en la velocidad que provoca falta de convergencia, excepto que controlemos esa velocidad. El método más usual es poner un límite máximo a esa velocidad para cada dimensión *D* y para cada partícula *i*.

$$\begin{gathered} \text{Si } v_{id} > V_{max} \text{ entonces } v_{id} = V_{max} \\ \text{Si no, si } v_{id} < -V_{max} \text{ entoces } v_{id} = -V_{max} \end{gathered} \tag{3.5}$$

El resultado de hacer esto es que la partícula oscile entre unos límites evitando que converja en un punto y a la vez que no se aleje demasiado. Algunos otros métodos de control son el concepto de "peso de inercia" o los "coeficientes de constricción" que se verán más adelante.

5.1.4. Algoritmo original del PSO

El algoritmo que se presenta a continuación es el algoritmo originario, a partir de este han surgido nuevas modificaciones y optimizaciones que serán o no beneficiosas dependiendo del problema que se nos presente.

1- Inicialización del array de partículas de población con posiciones y velocidades aleatorias para *D*-dimensiones en el espacio de búsqueda.

2- **Inicio del Bucle**

3- Para cada partícula se evalúa la función de optimización deseada en *D* dimensiones.

4- Se compara la evaluación de la posición partícula que hemos hecho con su $pbest$. Si la nueva es más favorable, se actualiza $pbest$ con el valor actual y p_i será igual a la localización actual x_i en el espacio de *D* dimensiones.

5- Se identifica la partícula con mejor posición del entorno encontrada hasta el momento y se asigna a p_g.

6- Cambiamos la velocidad y la posición de las partículas para ser evaluada en la próxima iteración siguiendo la siguiente ecuación (ver las anotaciones debajo):

$$\overrightarrow{v_i} \leftarrow \overrightarrow{v_i} + \vec{U}(0, \emptyset_1) * (\overrightarrow{p_i} - \overrightarrow{x_i}) + \vec{U}(0, \emptyset_2) * \left(\overrightarrow{p_g} - \overrightarrow{x_i}\right)$$
$$\overrightarrow{x_i} \leftarrow \overrightarrow{x_i} + \overrightarrow{v_i}$$

7- Si se cumple el criterio que establezcamos, normalmente un número máximo de iteraciones o una función lo suficientemente ajustada, saldremos del bucle.

8- **Fin del bucle**

Notas:

-$\vec{U}(0, \emptyset_i)$ representa un vector de número aleatorios uniformemente distribuidos comprendidos entre $(0, \emptyset_i)$ que son generados en cada iteración para cada partícula.

- En la versión originaria del PSO, cada componte $\vec{v_i}$ esta comprendida entre dos valores máximos y mínimos [-V_{max},+ V_{max}].

- **Pseudo código del PSO**

Creación e inicialización de la nube (matriz) S compuesta de N partículas, cada partícula con D dimensiones;

repetir

for *cada partícula desde i=1 hasta i=N* **do**

//evaluación de la mejor posición personal

if $f(x_i) < f(p_i)$ **then**

$p_i = x_i$;

end

//evaluación de la mejor posición global

if $f(p_i) < f(p_g)$ **then**

$p_g = p_i$;

end

end

for *cada partícula desde i=1 hasta i=N* **do**

actualización de la velocidad mediante la ecuación (3.3);

actualización de la posición mediante la ecuación (3.1);

end

hasta que se cumpla la condición de parada;

3.1.5. Algunos aspectos del algoritmo

Es interesante resaltar algunos aspectos básicos del método. Tales como comprender los componentes de la velocidad y como afecta cada uno a la velocidad final, dar una visión gráfica de cómo funciona la actualización de la velocidad y cómo se mueven las partículas en el espacio, las condiciones de inicialización de la propia velocidad y la posición además de ver cuales condiciones se pueden establecer como fin de bucle de iteraciones.

- **Componentes de la velocidad**

Como se ha visto, la ecuación de actualización de la velocidad (3.3) está formada por 3 componentes. Estos componentes van a ser: la velocidad previa, el componte individual o cognitivo y el componente global o social.

- La velocidad previa: representada como $\overrightarrow{v_i}$ es la velocidad con la que se movía la partícula hasta esta actualización. Se puede tomar este término como un momento de inercia, que hace que el cambio en la velocidad no sea tan drástico sino que tenga una ligera continuidad en el tiempo. A este elemento de le suele denominar inercia.
- El componente individual/cognitivo $\varphi_1 * (\overrightarrow{p_i} - \overrightarrow{x_i})$: mide el rendimiento de la partícula *i* en función de sus posiciones anteriores. Kennedy y Eberhart interpretaban este término como la "nostalgia" de la partícula, ya que tira de la partícula hacia los mejores anteriores encontrados por ella y no por el grupo.
- El componente global/social $\varphi_2 * \left(\overrightarrow{p_g} - \overrightarrow{x_i}\right)$: representa el peso que tiene el grupo de partículas. Hace que la partícula se dirija hacia el mejor punto encontrado entre todo el grupo.

Tanto el peso del componente global como el del social tienen un elemento aleatorio o estocástico φ que le da la capacidad de exploración. Más adelante se verá qué valor tomará.

- **El movimiento de las partículas en el plano**

Este apartado busca dar una vision gráfica de la dinámica de las partículas. Se ha optado por representar el movimiento en un plano de 2D, pero el número de dimensiones, como se ha visto, dependerá del problema.

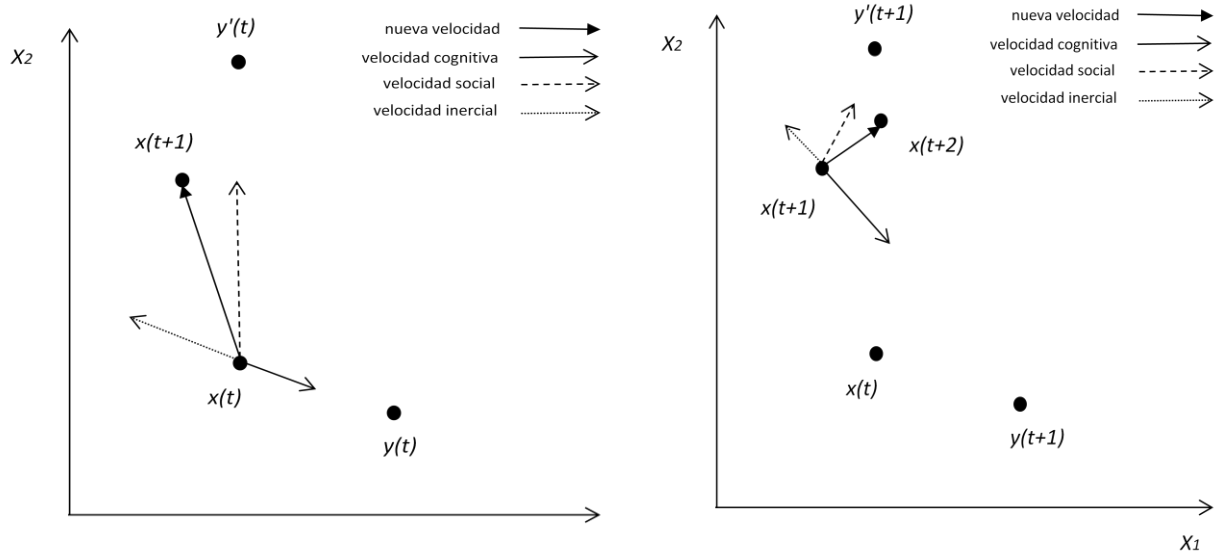

Figura 3.1: izqda.: primer movimiento de la partícula tras iteración en un plano de 2D; dcha.: segundo movimiento de la partícula tras la siguiente iteración.

La primera figura representa el estado de la partícula en un momento de tiempo *t*. Se puede apreciar como cada uno de los pesos tira de la partícula hacia sí. De ella tiran la inercia, la componente global y la componente individual. El peso de cada componente se representa con un vector mayor o menor, la suma de todos ellos, es decir, la ecuación de actualización de velocidad dará como resultado una nueva posición (*t+1*). Una vez localizada la partícula en la nueva posición se pueden dar dos situaciones:

- Que la nueva posición sea peor que la mejor global que teníamos hasta el momento, en cuyo caso en la próxima iteración la partícula seguirá teniendo en cuenta el mismo mejor global y buscara una nueva mejor posición.
- Que la nueva posición sea mejor que la mejor global que se había encontrado hasta el momento, en cuyo caso se actualizara la mejor global con la posición actual de la partícula y atraerá al resto de partículas de la nube hacia esta nueva posición mejor.

- **Inicialización de posición y velocidad**

El primer paso en el algoritmo del PSO es inicializar los parámetros de control de la nube. Se deben establecer las velocidades iniciales, las posiciones iniciales así como algunas constantes (estas últimas se establecerán en el apartado 3.3).

Es muy importante encontrar un buen criterio de inicialización de los anteriores parámetros,

ya que esto nos marcara la evolución posterior del algoritmo y por ende el resultado final que conseguiremos.

La posición de las partículas normalmente se suele distribuir uniformemente sobre el espacio de búsqueda, ya que de no ser así a las partículas les va a costar llegar hacia zonas distantes, siendo propensas a caer en mínimos locales.
Para establecer esta posición tendremos unos vectores con valores máximos y mínimos de localización en cada dimensión x_{max} , x_{min} . Una posible inicialización podría ser la siguiente [19]:

$$x_i(0) = x_{min,j} + U(0,1)(x_{max,j} - x_{min,j}), \quad \forall j = 1, \dots D, \quad \forall i = 1, \dots N \tag{3.6}$$

Respecto a la velocidad podemos simplemente inicializarla a cero que se presenta como la solución más fácil. Otra opción es darle un valor aleatorio, aunque no es del todo recomendable además de que de hacerlo tiene que ser con sumo cuidado. Físicamente tiene más sentido la primera opción ya que lo normal es considerar que si situamos partículas en un espacio estas comiencen a moverse desde el estado de reposo y no que se sitúen en el espacio en movimiento. En cualquier caso, si se decide localizar las partículas con una velocidad aleatoria ésta no debe de ser excesivamente alta, ya que esto implicaría introducir una gran inercia para movimientos posteriores lo que dificultaría la convergencia, alargando el tiempo de ejecución del algoritmo. Es por esto que a priori lo más fácil sería establecer.

$$v_i = 0 \tag{3.7}$$

Otro factor a establecer desde el inicio es la mejor posición de la partícula p_i . Esta se inicializará con la primera posición de la partícula.

$$p_i(0) = x_i(0) \tag{3.8}$$

Por último, en el algoritmo del PSO se deben establecer las condiciones de parada del bucle principal. Hay dos reglas claras que las condiciones de parada que se establezcan deben evitar: que el algoritmo converja antes de que se haya encontrado un optimo lo suficientemente bueno y evitar excesivos cálculos de la función “fitness” que prolonguen

demasiado el procesamiento computacional del algoritmo.
Las condiciones de parada más habituales son las siguientes:

-Finalizar cuando se alcance un número máximo de iteraciones. El número de iteraciones máximo debe ser elegido cuidadosamente, ya que si es bajo la solución puede no estar lo suficientemente ajustada y si es excesivamente alto resultara en demasiado tiempo de computación. Por esto se suele acompañar de un criterio de convergencia de forma que si tras una serie de iteraciones no converge, el algoritmo pare.

-Finalizar cuando se alcance una solución lo suficientemente buena. Finalizara la iteración cuando se llegue a un error aceptable. Si llamamos x^* a la solución buscada, x_i a la solución de una partícula y $\in$ al error, la búsqueda finalizará si se cumple $f(x_i) \leq |f(x^*) - \in|$ [19].
El error ha de ser elegido adecuadamente.
Si es demasiado pequeño va llevar demasiado tiempo la computación o incluso puede llegar a ser una solución inalcanzable. De ser demasiado grande producirá soluciones no suficientemente buenas, quedándose en mínimos locales. Además este método no asegura por si mismo que el algoritmo haya convergido.

-Finalizar cuando no se observa mejora tras una serie de iteraciones. Existen diversas formas de determinar el rendimiento. Una de ellas es ver si la media de las variaciones de la posición de la partícula tras una serie de iteraciones es muy pequeña, significara que la nube de partículas ha convergido. Otra puede ser ver una media de la variación de velocidad muy baja en una serie de iteraciones que determina que la búsqueda puede parar. También podría finalizar la búsqueda si no ha habido mejora en un número determinado de iteraciones.

-Finalizar cuando el pendiente de la función objetivo es aproximadamente cero [20]. También podemos basar la salida del bucle principal en función de la pendiente que alcancen las partículas en el espacio de búsqueda. Si establecemos el siguiente ratio:

$$f' = \frac{f(p_g(t)) - f(p_g(t-1))}{f(p_g(t))} \tag{3.9}$$

En el caso de que $f' < \in$ durante una serie determinada de iteraciones se considerara que la nube ha convergido. Es una opción muy sólida en la que se ve si la nube va realmente

progresando en el espacio de búsqueda. Como problema principal se encuentra el peligro de caer en mínimos locales.

3.2. Modificaciones básicas del PSO

A partir del algoritmo básico del PSO han ido surgiendo una serie de modificaciones que dan más solidez al método, permitiendo un uso más fiable en problemas de optimización en los que es difícil encontrar una convergencia hacia buenas soluciones. Entre estas modificaciones encontramos la introducción de un peso de la inercia, control de la velocidad así como de distintos modelos de la misma y de distintas formas de encontrar la mejor posición personal y global de las partículas.

3.2.1. Control de la velocidad

Hay dos importantes características a diferenciar en la búsqueda de una solución óptima en el espacio, la exploración y explotación. Podemos definir exploración como la capacidad de las partículas de explorar diferentes regiones buscando la óptima mientas que la explotación es la habilidad de buscar en un área determinada y prometedora la mejor solución. Es por esto que un buen algoritmo debe explotar y explorar equilibradamente. Esta capacidad se controlara con la velocidad de las partículas. La primera medida para controlar la exploración global es mantener la velocidad de las partículas entre cierto rango [21], con un valor máximo que evite grandes saltos en el espacio de búsqueda, de forma que si la nueva velocidad de la partícula fuera mayor que la establecida como máximo esta será el valor máximo directamente. Comprendiendo $V_{max,j}$ como la velocidad máxima que se puede alcanzar para la dimensión j tenemos que:

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t+1)\ si\ v'_{ij}(t+1) < V_{max,j} \\ V_{max,j}\ si\ v'_{ij}(t+1) \geq V_{max,j} \end{cases} \quad (3.10)$$

Donde $v'_{ij}(t+1)$ es la nueva velocidad tras la iteración.

El valor que se le dé a la velocidad máxima es muy importante ya que mayores velocidades permiten una mayor explotación global mientras que velocidades menores implican una búsqueda más local. Si la velocidad es demasiado alta podemos encontrar que la nube pase

por alto buenas regiones que debieran ser explotadas localmente, mientras que velocidades demasiado bajas pueden centrar demasiado su búsqueda en mínimos locales pudiendo no ser estos los más favorables, además de alargar demasiado el algoritmo debido a cambios muy pequeños en la posición. Es por esto que encontrar un valor adecuado de $V_{max,j}$ se hace difícil y variará en función del problema. Habitualmente se establecerá de la siguiente manera:

$$V_{max,j} = \delta(x_{max,j} - x_{min,j}) \tag{3.11}$$

Donde $x_{max,j}$ y $x_{min,j}$ son los valores máximos y mínimos de la posición en la dimensión j y δ está comprendida entre los valores (0,1].

Hay una serie de problemas que no se resolverán con el control de la velocidad y otros que pueden aparecer a raíz de este control. Se ha de tener en cuenta que este control de la velocidad no reducirá únicamente el módulo del vector velocidad resultante sino que también modificara su dirección debido a que al modificar los módulos de los vectores en cada dimensión, el resultante modificara su dirección final. Otro problema que aparecerá será la dificultad de explotar regiones locales debido a que podrá llegar el momento en el que se bordee la solución óptima sin llegar a ella, encontrándonos con un fallo de explotación. Este problema se podrá solventar con la introducción del concepto de inercia, del que se hablará en el siguiente punto. Otra forma de solventarlo será reducir periódicamente el límite de la actualización de la velocidad. La idea de este concepto será comenzar con límites mas holgados favoreciendo la exploración y progresivamente ir reduciendo los límites para concentrar la búsqueda en áreas locales motivando de esta forma que la nube se acerque al mínimo requerido.
Los dos métodos más utilizados para aplicar la reducción periódica de velocidad son los siguientes:

- Cambiar el límite de velocidad máxima si tras un número τ determinado de iteraciones [22] no se ha encontrado una mejor posición global. El límite de velocidad máxima quedaría de la siguiente forma:

$$V_{max,j}(t+1) = \begin{cases} \beta V_{max,j}(t+1) \; si \; f(pg(t)) \geq f(pg(t-t')) \;\; \forall \, t' = 1, \ldots, \tau \\ V_{max,j}\,(t) \;\; en \; el \; otro \; caso \end{cases} \tag{3.12}$$

Donde β comenzara con un valor de 1 e ira progresivamente disminuyendo hasta 0.01.

- Reducir exponencialmente la velocidad máxima siguiendo la siguiente fórmula [23]:

$$V_{max,j}(t+1) = (1 - (\frac{t}{n_t})^{\alpha})V_{max,j}\,(t) \tag{3.13}$$

Donde α es positiva y constante, tomada por prueba y error y n_t el número de iteraciones máximo.

3.2.2. Peso de la inercia

Este concepto, que fue introducido por Eberhart y Shi [24] tenía como objetivo eliminar la necesidad de controlar la velocidad como se ha visto anteriormente. Sin embargo, en la práctica, es posible que se consiga un resultado más robusto combinando las dos técnicas.
La idea es controlar el momento de la partícula dándole mayor o menor peso a la velocidad anterior. Esto modificaría la ecuación de actualización de velocidades vista anteriormente, quedando de esta forma:

$$\overrightarrow{v_i} = w\overrightarrow{v_i} + \varphi_1(\overrightarrow{p_i} - \overrightarrow{x_i}) + \varphi_2(\overrightarrow{p_g} - \overrightarrow{x_i}) \tag{3.14}$$

Si se toman valores de $w \geq 1$ la velocidad se irá incrementando en cada iteración acercándose a la velocidad máxima, siempre y cuando se haya tomado también un control en la velocidad (explicada anteriormente). Por otro lado para valores de $w<1$ la velocidad disminuirá progresivamente hasta llegar a cero. Por esto para valores altos se motiva la exploración de mínimos globales y para valores más pequeños la exploración de mínimos locales.
Como ocurría con el control de la velocidad, la elección del valor de w dependerá de cada problema. Al igual que mediante el control de la velocidad, el método que se suele seguir es variar su valor con el tiempo, comenzando con valores altos favoreciendo la búsqueda general e ir disminuyendo el valor para favorecer la búsqueda local.
Algunos métodos que se siguen para variar la inercia con el tiempo son las siguientes:

- Ajuste aleatorio: en cada iteración la inercia se toma aleatoriamente siguiendo una distribución Gaussiana:

$$w \sim N(0.72, \sigma) \tag{3.15}$$

Tomando σ un valor que haga que no predominen valores mayores que 1.

- Decrecimiento lineal: propuesto por Naka *et al.* [25] en los que se suele comenzar con un valor de inercia alto (comúnmente 0.9) y va disminuyendo progresivamente (comúnmente hasta 0.4) siguiendo la siguiente fórmula.

$$w(t) = \big(w(0) - w(n_t)\big)\frac{(n_t - t)}{n_t} + w(n_t) \tag{3.16}$$

Donde n_t es el número máximo de iteraciones, $w(n_t)$ es el valor final que toma la inercia, $w(0)$ es el valor inicial y $w(t)$ es el valor que toma la inercia en el momento t.

- Decrecimiento no lineal: de la misma forma, la idea es iniciar con un valor alto que irá disminuyendo, en este caso, de forma no lineal. Mediante el decrecimiento no lineal el tiempo aplicado a la exploración es más corto que mediante decrecimiento lineal y el tiempo dedicado a la explotación, sin embargo, es mayor que con el mismo. Los métodos de decrecimiento no lineal más utilizados son:

-El propuesto por Naka *et al.* [25]:

$$w(t+1) = \frac{(w(t) - 0.4)(n_t - t)}{n_t + 0.4} \tag{3.17}$$

con un valor de $w(0)$=0.9.

-El propuesto por Sobieszczanski-Sobieski y Venter [26]:

$$w(t+1) = \alpha w(t') \tag{3.18}$$

con un valor de α=0.975, siendo t' el momento en el cual la inercia ha cambiado. Este cambio de inercia se dará en el caso en el que el cambio en la función fitness no haya sido lo suficientemente grande. Para determinar si el cambio es satisfactorio se suele evaluar un porcentaje determinado de partículas comprobando si su variación ha sido

muy baja, de ser así se modificaría el valor de la inercia. En este caso los valores inercia inicial y mínima posible final que se toman son $w(0) = 1.4$ y $w(n_t) = 0.35$.

-El propuesto por Clerc [27], el cual propone un método del que asegura que en pocas iteraciones se obtiene un gran resultado. Una inercia adaptativa cuyo valor cambiará proporcionalmente a la mejora que se haya producido en la nube. Para ello cada partícula tendrá su valor específico de inercia basándose en su distancia al mejor local (p_i). El valor de inercia se modificara así:

$$w(t+1) = w(0) + (w(n_t) - w(0))\frac{e^{m_i(t)} - 1}{e^{m_i(t)} + 1} \tag{3.19}$$

donde la mejora relativa, m_i se toma como

$$m_i(t) = \frac{f\big(p_i(t)\big) - f\big(x_i(t)\big)}{f\big(p_i(t)\big) + f\big(x_i(t)\big)} \tag{3.20}$$

con un valor final de inercia aproximado de $w(n_t) \approx 0.5$ y un valor inicial $w(0) < 1$. Además se puede tomar la mejor posición global (p_g) en vez de la local (p_i) en el caso de que se le quiera dar más peso a la posición global.

3.2.3. Coeficientes de constricción

El método de coeficiente de constricción desarrollado por Clerc [28] cuya finalidad es, de igual manera que en los anteriores, controlar el peso entre exploración y explotación, fue llamado "Tipo1" en su versión más simple. La razón por la que se desarrollo era encontrar un método que no buscase medidas "paliativas" para evitar que la solución divergiera, como por ejemplo mediante los métodos de control de velocidad, sino un algoritmo que demostrase de manera coherente la convergencia en la solución. Para ello se incluirá de manera similar a la inercia el término χ, quedando la actualización de velocidad:

$$\overrightarrow{v_i} = \chi\big[\overrightarrow{v_i} + \varphi_1(\overrightarrow{p_i} - \overrightarrow{x_i}) + \varphi_2(\overrightarrow{p_g} - \overrightarrow{x_i})\big] \tag{3.21}$$

donde $\varphi = \varphi_1 + \varphi_2 \geq 4$ y

$$\chi = \frac{2}{\varphi - 2 + \sqrt{\varphi^2 - 4\varphi}} \tag{3.22}$$

Bajo este método se suelen tomar valores de $\varphi = 4.1$, $\varphi_1 = \varphi_2$, y el valor del término χ=0.7298. Este valor para los términos resultaría en la velocidad previa multiplicada por 0.7298 y los términos de incremento en la posición multiplicados por $0.7298 * 2.05 \approx 1.496$. Por lo tanto se puede interpretar de forma equivalente al método de peso inercia, tomando los valores $w = \chi$ y $\varphi_i = \chi\varphi_i$. Por lo que $w = 0.7298$ y $\varphi_1 = \varphi_2 = 1.4961$. Si comparamos los métodos de constricción y peso de inercia vemos que los dos tienen como objetivo converger en un punto óptimo. En principio no sería necesario añadir un control de la velocidad sin embargo en experimentos desarrollados por Eberhart y Shi [29] comprueba que añadiendo un control en la velocidad se consiguen mejores resultados. De esta forma se consigue un algoritmo de optimización independiente del problema al que nos enfrentemos, por lo que puede considerarse un método robusto y versátil.

3.3. Parámetros del PSO

El algoritmo del PSO tiene una serie de elementos que deben ser fijados a priori tales como, la cantidad de partículas que van a actuar n_t , los elementos φ_1y φ_2 (también llamados coeficientes de aceleración) cuyo valor dará más o menos peso a la mejor posición global o local y el número de iteraciones necesario para llegar a la solución óptima.

- **Tamaño de la nube**

Es el número de partículas que van a formar parte de la nube, n_t. El número de las mismas dependerá del problema, para problemas sencillos harán falta menos partículas para encontrar una solución óptima, mientras que para problemas más complicados harán falta más partículas. La elección del número de partículas debe ser lo más ajustado posible al problema ya que si bien es cierto que un mayor número de partículas facilitara encontrar valores más óptimos también nos incrementara el tiempo de computación de nuestro algoritmo. Si bien estudios [30] determinan que sería suficiente un numero entre 10 y 30 partículas para encontrar la solución, estudios posteriores como el de R.Poli *et al*. [31] recomiendan una

horquilla de entre 20-50 partículas ya que como se ha dicho antes, dependerá del problema.

- **Número de iteraciones**

Dependerá de la complejidad del problema, a mayor numero de iteraciones mayor probabilidad de encontrar una mejor solución pero también mayor tiempo computacional.

- **Coeficientes $\boldsymbol{\varphi_1}$y $\boldsymbol{\varphi_2}$**

También llamados coeficientes de aceleración. El valor de los mismos es de gran importancia ya que le darán mayor o menor peso tanto a la mejor posición encontrada por cada partícula individual como a la mejor global encontrada por la nube. En [31] las componentes $\varphi_1(\overrightarrow{p_i} - \overrightarrow{x_i})$ y $\varphi_2(\overrightarrow{p_g} - \overrightarrow{x_i})$ son interpretadas como las fuerzas de atracción producidas en un muelle con coeficientes de rigidez $\varphi_1/2$ y $\varphi_2/2$. Con esta interpretación los coeficientes representarían las constantes elásticas de los muelles tirando de la partícula, por lo que se puede ver que calibrando bien los mismos lograremos un equilibrio entre explotación y exploración. El óptimo valor de los mismos ha sido objeto de muchas modificaciones, siendo el originario φ_1=φ_2=2 sin embargo, estudios posteriores han logrado optimizaciones como las anteriormente vistas: "peso de la inercia" y "coeficientes de constricción", que modifican su valor y gracias a los cuales se logran grandes resultados.

Capítulo 4

Scan matching con PSO

4.1. Introducción al algoritmo

En este capítulo se va a exponer el proceso que se ha seguido para implementar el scan matching siguiendo el algoritmo del PSO (Particle Swarm Optimization) y utilizando las propiedades del color.

En este capítulo no se va a desarrollar el área que tiene que ver con las propiedades del color de forma amplia ya que no es competencia del presente proyecto, si se desea ampliar la información, el lector puede consultar el trabajo desarrollado por Martín *et al.* [5].

El objetivo que persigue el scan matching es minimizar el error que se presenta al localizar un robot en un entorno mediante métodos odométricos. Para ello se buscará encontrar la correspondencia entre dos escaneos del entorno tomados por el robot.

En este caso se ha seguido un método de scan matching global, esto es, se buscará los puntos de correspondencia en el modelo para cada punto de los datos y se calculará la mínima distancia entre ellos.

El proceso tomará por tanto 2 conjuntos de puntos en 3D, modelo y datos, y buscara la transformación, entendida como rotación y translación, que aplicada sobre la serie de datos maximice la correlación con el modelo. La correlación entre las dos series de datos se buscara de manera estocástica mediante un algoritmo evolutivo. El algoritmo iterativamente evaluara, basándose en una función de coste, si la transformación que se ha aplicado en esa iteración favorece la correlación entre las 2 series de datos. La solución está representada por la

posición del robot, algo que le diferencia del ICP clásico.

El algoritmo evolutivo que se ha seguido para conseguir la correlación es el PSO. Este algoritmo se trata de forma genérica y amplia en el capítulo 5, de modo que en este apartado se presentara estrictamente cómo se ha implementado para solucionar este problema en concreto. Las ideas principales del método que se ha seguido se pueden encontrar en *Algoritmo1*. Se puede ver además en la figura 4.1 el flujograma general del proceso que se ha seguido. Este proceso se puede dividir en 2 fases principales:

- Fase de preprocesamiento de las imágenes: en esta parte se analizan las 2 imágenes que se van a hacer correlacionar y se filtran según diferencias de ΔE entre los puntos con el objetivo de seleccionar puntos característicos. Esta fase es de gran importancia ya que se consigue que el coste computacional final sea menor manteniendo la precisión en la convergencia, como se puede ver en la investigación de Martín *et al*.
- Fase de correlación: en esta parte se integra el algoritmo propio del PSO para solucionar el problema de scan matching entre los pares de imágenes previamente filtradas. Para ello se tratara de minimizar la distancia entre los puntos característicos de forma que se logre la máxima correlación entre los mismos. Esta parte es la que se desarrollara en este capítulo.

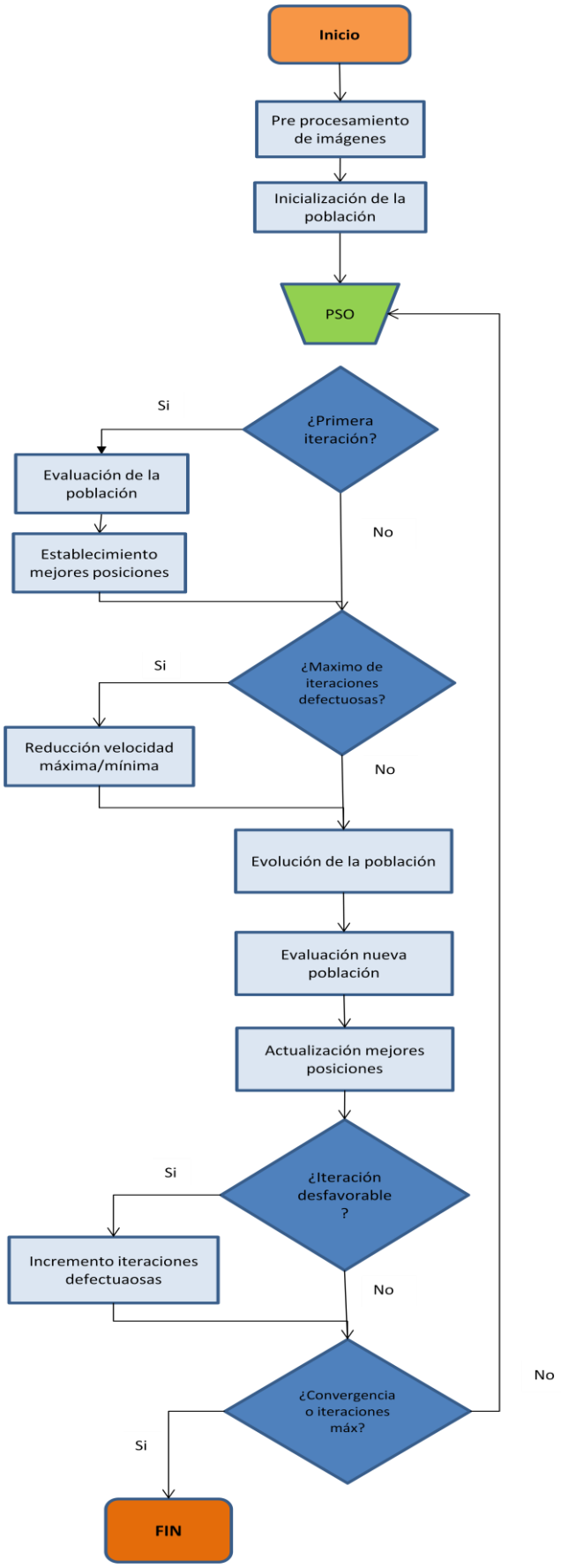

Figura 4.1: Flujograma general del algoritmo de scan matching mediante PSO

4.2. Implementación del algoritmo

La solución se va a buscar con un conjunto de partículas que representaran una población de *Np* elementos. Cada uno de estos individuos tendrá una posición en el espacio y constaran de 6 elementos debido a que el robot tiene 6 grados de libertad de las cuales 3 son la posición *x, y, z* en coordenadas cartesianas y las otras 3 representaran la orientación $\varphi, \phi\ y\ \psi$ según los ángulos de Euler.

$$pob_i^k = (x_i^k, y_i^k, z_i^k, \varphi_i^k, \phi_i^k, \psi_i^k) \tag{4.1}$$

Donde pob representa a cada elemento i de la población en la iteración k. La población de partículas iniciales se creará en el entorno de la posición dada por odometria siendo el objetivo reducir su error.

El concepto característico que se va a utilizar para buscar la correspondencia entre puntos del modelo y los datos será la transición de colores entre los mismos, es decir, se buscaran puntos con los mismos atributos de color y una vez localizados como iguales se buscara minimizar una función de coste que mide la distancia entre los mismos. La transición de color se obtiene en la *línea 1*.

La creación de la población comienza en la *línea 2* y consiste en una matriz de filas igual al número de partículas *Np,* en este caso se ha decidido tomar un tamaño de 50 partículas siguiendo la recomendación de R.Poli, J.Kennedy y T.Blackwell [31], y de columnas igual a 7: en la primera columna se asignara el error de coste de la partícula en esa posición y las 6 siguientes serán las correspondientes a posición y orientación. La matriz población quedaría de la siguiente forma:

$$\begin{bmatrix} e_i & x_i & y_i & z_i & \varphi_i & \phi_i & \psi_i \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ e_{Np} & x_{Np} & y_{Np} & z_{Np} & \varphi_{Np} & \phi_{Np} & \psi_{Np} \end{bmatrix} \tag{4.2}$$

En la *línea 5* comienza el bucle principal de iteraciones que buscara posibles soluciones hasta llegar a la convergencia o hasta un número máximo establecido de iteraciones. Se considerara que el algoritmo ha llegado a la convergencia si durante un número determinado de iteraciones la diferencia entre costes es menor a 0.0005, que significará que ha caído en un mínimo y será tomado como solución final.

Al comienzo de la primera iteración, *línea 6*, se calculará el coste de la población inicial que se ha creado. Para ello se creará una matriz auxiliar que corregirá la posición de la los datos acorde con las posiciones de las partículas de la población. Esta matriz que contiene los datos corregidos en posición es la que se enviará a nuestra función de cálculo de coste junto con el modelo. Tras el cálculo de los costes para cada posición de cada partícula estos se incluirán en la primera columna de la matriz población.

La fórmula de cálculo de coste a minimizar es la derivada de la distancia entre los puntos de correspondencia de las transiciones de color:

$$e = \sum_{c=1}^{C} d(m_{ic}, d_{jc})^2 \tag{4.3}$$

Donde *C* representa el número de correspondencias y $d(m_{ic}, d_{jc})$ la distancia entre dos puntos de correspondencia.

Una vez se ha calculado el coste y se ha incluido en la matriz población esta se ordenará (*línea 10*) de forma que en la primera fila aparezca la partícula de la nube con el menor coste. Se considerará entonces (*línea 11*) la primera fila como la partícula de la población con mejor resultado llamada *pob_best*. Además se creara una nueva variable que representara la mejor de las soluciones encontradas hasta el momento entre todas las iteraciones, se llamara *pob_best_glob*. En su inicialización (*línea 12*) esta tomara el valor de *pob_best*.

En el bucle de la *línea 14* se introduce el concepto de velocidad máxima y mínima. Estos valores están relacionados con la evolución de la posición de las partículas que se realizara posteriormente. Las partículas cambiaran su posición siguiendo un cambio en su velocidad. En este bucle se considera que si la iteración ha sido desfavorable en un número determinado de veces las partículas se estarían alejando del valor mínimo tras su modificación, por lo que, para controlar los saltos en su posición, se reducirán sus valores máximos de velocidad alcanzables. El método que se ha seguido para reducir progresivamente los valores máximos de velocidad en caso necesario [22]:

$$V_{max}(t+1) = \begin{cases} \beta V_{max}(t+1) \; si \; f(pob\_best\_glob(t)) \geq f(pob\_best\_glob(t-t')) \;\; \forall\, t' = 1, \ldots, \tau \\ V_{max}\,(t) \;\; en \; el \; otro \; caso \end{cases} \tag{4.5}$$

Donde τ es el número máximo de iteraciones desfavorables, en este caso fijada a 2 y β comenzara con un valor de 1 e irá disminuyendo hasta un valor mínimo de 0.01.

En la *línea 18* comienza el bucle evolutivo de las partículas. Este bucle producirá un cambio

en el posicionamiento de las partículas con el objetivo de acercarse a una mejor solución. El cambio en la posición se realizara a través de una modificación de la velocidad de movimiento de las mismas en cada una de sus componentes de posición (*x, y, z*) y orientación (φ, ϕ, ψ). Para desarrollar la modificación en la velocidad se han aplicado técnicas de optimización como el de "peso de la inercia" propuesto por Eberthart y Shi [24] y los "coeficientes de constricción" desarrollado por Clerc [28]. Estas optimizaciones están explicadas individualmente de manera amplia en el capítulo 3, por lo que en este caso se va a explicar el resultado final. La velocidad quedará de la siguiente forma:

$$v = (w * v) + \left(c * r_1 * \left(pob\_best(1,j) - pob(i,j)\right)\right) + (c * r_2 * \left(pob\_best\_glob(1,j) - pob(i,j)\right)) \quad (4.6)$$

Donde *i* es cada partícula, *j* cada una de sus coordenadas de posicionamiento, *w* la inercia con un valor constante de 0.729, *c* una constante de valor 1.49 y $r_{1,2}$ es un valor aleatorio comprendido entre [0,1].

La componente $\left(c * r_1 * \left(pob\_best(1,j) - pob(i,j)\right)\right)$ será la que haga a la partícula tender a ir hacia mejores posiciones locales mientras que la componente $(c * r_2 * \left(pob\_best\_glob(1,j) - pobi,j\right)$ provocará el acercamiento hacia mejores posiciones globales. De esta forma se consigue una evolución en la velocidad equilibrada entre búsqueda local y global. El proceso evolutivo finalizara aplicando la nueva velocidad a las partículas de la población:

$$pob\ (i,j) = pob\ (i,j) + v \quad (4.7)$$

Tras la evolución de las partículas se estudiará la nueva solución calculando la función de coste para esa nueva posición de manera similar a la descrita anteriormente, *linea.25*. Una vez calculado el nuevo coste se analizara si la solución propuesta es más favorable que las registradas hasta el momento. De nuevo la matriz de partículas se ordenará de forma descendente en función del coste asociado a cada elemento. La mejor solución del grupo se asignará a *pob_best*, *línea 29*. El siguiente paso (*línea 30*) será comprobar si la *pop_best* en esa iteración es mejor que *pop_best_glob* guardada hasta el momento, de ser así se asignara a esta su valor y el contador de iteraciones fallidas *cont_fail_it* se reseteará, de no serlo el contador de iteraciones se incrementará.

En la línea 36 se integra la prueba de posible convergencia. En este punto se comprueban los

costes resultantes de la iteración anterior y la actual y si se considera que la diferencia entre los mismos durante un número determinado de iteraciones es menor que 0.0005 se llegará a la conclusión de que se ha llegado a la convergencia y se producirá la salida del bucle considerando esta última como la mejor solución encontrada.

Por último se mostraran por pantalla elementos como los datos de la mejor solución en esa iteración y de las iteraciones registradas hasta el momento. Además se mostraran las imágenes del matching resultante entre los datos y el modelo.

Algoritmo1 Scan Matching basado en el PSO usando las Propiedades del Color

modelo_tc,datos_tc ← *extraer_tc (modelo,datos)* ▫Extracción de transición del color
for *i = 1 : Np* **do**
    $pob_i^1 \leftarrow inic\_pob(posic\_datos\_inicial)$ ▫Inicialización de la población
end for
while *(condición_parada)* ▫ Bucle principal de iteraciones
    if (*primera_iteracion*)
        for i = 1 : N_p
            $e \leftarrow coste\ (modelo\_tc, datos\_tc, pob)$ ▫ Función de cálculo del coste
        end for
        $pob\_sort \leftarrow$ *orden (pob)* ▫ Ordenación de la población
        pob_best ← pob_sort_1 ▫ Mejor solución de la población
        pop_best_glob ← *pop_best*
    end if
    if *(iteración_desfavorable* ≥ *n)* ▫ Reducción de velocidad de las partículas
        Vmax = Vmax_menor
        Vmin = Vmin_menor
    end if
    for *i = 1 : Np do* ▫ Evolución de la población
        for *j = 1 : D*
            $V = [V\_random]_{Vmin}^{Vmax}$
            $V = W * V + \varphi_1(p_i - x_i) + \varphi_2(p_g - x_i)$
            $pob_{i,j} = pob_{i,j} + V$
        end for
    end for
    for i = 1 : N_p
        $e \leftarrow coste\ (modelo\_tc, datos\_tc, pob)$ ▫ Función de cálculo del coste
    end for
    $pob\_sort \leftarrow$ *orden (pob)* ▫ Ordenación de la población
    pob_best ← pob_sort_1 ▫ Mejor solución de la población
    if *(e_pob_best < e_pob_best_glob)* ▫ Actualización de mejor solución global
        pob_best_glob = pob_best
        n = n_reset
    else
        n = n_nueva
    end if
    if (*mínima diferencia de costes anterior/actual*) ▫Prueba de convergencia

```
37:      conv= conv_nueva
38:    else
39:      conv= conv_reset
40:    end if
41:    bestmem ← pop_best_glob                    ▫ Mejor solución registrada
42: end while
```

Capítulo 5

Experimentos y resultado

Para realizar los experimentos, el algoritmo se ha implementado mediante la herramienta Matlab. Se ha utilizado además el data set *Freiburg2* (tomado por Jürgen Sturm) con una cámara Kinect montada en un robot Pioneer. Se han tomado además el preprocesamiento de imágenes y la función de coste del trabajo realizado por Martín *et al*. [5]. Los experimentos han sido realizados según la siguiente configuración de preprocesamiento de imágenes y función de coste:

5.1. Parámetros para el preprocesamiento de imágenes

En el presente proyecto se ha utilizado un método de preprocesamiento de imágenes. Según el estudio realizado por Martín *et al.* [5] las propiedades del color pueden ser una herramienta muy potente a la hora de considerar los puntos a correlacionar en un proceso de scan matching.

En esta fase se hace una extracción de puntos característicos basados en la divergencia de ΔE, tras este proceso, solo se tomaran los puntos con diferencias en color para realizar el scan matching, lo que ahorrará coste computacional. La elección de 2 valores se hacen necesarios para establecer qué puntos seleccionar, estos son: el valor mínimo de ΔE , que considerará puntos con distinto color, y un radio de muestreo que servirá también de delimitador para la elección de puntos con distintas características.

Según los experimentos realizados por Martín *et al.* [5] con valores de ΔE y radio de muestro de 10 y 0.1 respectivamente se obtienen buenos resultados, logrando llegar a la convergencia manteniendo la precisión y ahorrando coste computacional. Esos serán los valores utilizados para la fase de preprocesamiento de imágenes.

5.2. Parámetros en el PSO

Existen una serie de parámetros que se han de configurar para sacar el máximo partido al PSO. Aunque se hayan estimado valores generales para esos parámetros que sirven para una gran parte de los problemas de optimización con PSO, la elección de estos siempre va a estar sujeta al problema en particular al que nos enfrentemos. Es por esto que se ha querido buscar el valor óptimo para algunos de esos parámetros en la resolución del scan matching para este caso.

Lo que se buscará con la elección del valor de estos parámetros será el ahorro en coste computacional sin renunciar a la precisión del método, uno de los principales objetivos generales en los procesos de mapeado. En concreto se va a hacer un estudio del tamaño óptimo de la nube de partículas *Np* y de los valores máximo V_{max} y mínimo V_{min} de la velocidad alcanzable por las partículas.

Para los experimentos, en primer lugar se va a analizar el tamaño más adecuado de la nube de partículas para valores máximos y mínimos de velocidad de [1,-1] como referencia. Una vez obtenido el valor deseable de partículas en la nube se utilizará ese valor para calcular el óptimo de velocidades máximas y mínimas.

La elección de los pares de *frames* a analizar serán los mismos para todos los experimentos, incluido el análisis del valle de convergencia ya que más tarde se compararan los resultados obtenidos frente al método implementado mediante DE (Differential Evolution) [5].

5.2.1. Tamaño de la nube

Para la elección del tamaño de la nube se parte de los estudios que recomiendan tamaños desde 10 a 50 partículas [30,31]. Por esto se analizarán como posibles opciones nubes de 10,

30 y 50 partículas y se compararán los resultados según: correlación o no de los escaneos, mínimo de iteraciones necesarias para su convergencia, tiempo de convergencia y error en el coste.

Tabla 5.1: Resultados en el análisis de los 4 pares de escaneos para distintos tamaños de nube de partículas, según matching, iteraciones, tiempo y error de coste.

Números de escaneo: 1000-1010				
nº Partículas	Matching	Iteraciones	Tiempo	Error del coste
10	no	14	3,1s	0,64
30	si	75	30,3s	0,24
50	si	32	23,65s	0,24
Números de escaneo: 50-60				
nº Partículas	Matching	Iteraciones	Tiempo	Error del coste
10	no	38	9,32s	0,17
30	no	44	29,5s	0,16
50	si	33	35,7s	0,16
Números de escaneo: 1450-1451				
nº Partículas	Matching	Iteraciones	Tiempo	Error del coste
10	no	63	11,11s	0,72
30	si	45	24,5s	0,33
50	si	43	36,9s	0,33
Números de escaneo: 500-501				
nº Partículas	Matching	Iteraciones	Tiempo	Error del coste
10	no	55	8,5s	0,41
30	si	74	30,7s	0,1
50	si	40	28,6s	0,08
Resumen de datos				
nº Partículas	Matching	Iteraciones	Tiempo	Error del coste
10	0	42	8s	0,48
30	75	59	28,75s	0,2
50	100	37	31,2s	0,2

Como se puede observar en el resumen de la tabla 5.1, nubes de entre 30 y 50 partículas son las adecuadas ya que presentan altas proporciones de matching, como proponían Poli *et al*. [31].

Para este problema se han tomado nubes de 50 partículas ya que han resuelto el matching en todos los experimentos (ver resumen de datos de tabla 5.1), han necesitado menos iteraciones que las nubes de 30 elementos y el tiempo de ejecución está solo 2 segundos por encima del de las nubes de 30.

5.2.2. Valor de V_{max} y V_{min}

Para calcular los valores de velocidad máximos y mínimos se ha tomado una nube de partículas de 50 elementos que, como se ha visto en el anterior experimento es el valor más adecuado.

En principio, según estudios como el de *Poli et al.* [31] el uso de límites en las velocidades alcanzables por las partículas no sería necesario para llegar a converger en el caso de aplicar *coeficientes de constricción*, sin embargo, si queremos optimizar el proceso lo máximo posible se pueden añadir. En este caso se van a comprobar límites bajos, que son los que en general han dado buen resultado. Se comprobaran valores de 0.5, 1 y 2.

Tabla 5.2 : Resultados obtenidos para los 4 pares de escaneos según matching, iteraciones, tiempo y error de coste, en función de la velocidad máxima y mínima de las partículas.

Números de escaneo: 1000-1010				
[vmax,vmin]	Matching	Iteraciones	Tiempo	Error del coste
[0,5;-0,5]	si	34	24,5s	0,24
[1;-1]	si	33	24,78s	0,241
[2;-2]	si	36	28,02s	0,245
Números de escaneo: 50-60				
[vmax,vmin]	Matching	Iteraciones	Tiempo	Error del coste
[0,5;-0,5]	si	52	55,94s	0,16
[1;-1]	si	58	57,87s	0,165
[2;-2]	si	31	42,1s	0,167
Números de escaneo: 1450-1451				
[vmax,vmin]	Matching	Iteraciones	Tiempo	Error del coste
[0,5;-0,5]	si	57	43,1s	0,33
[1;-1]	si	25	24,89s	0,33
[2;-2]	si	35	32,96s	0,337
Números de escaneo: 500-501				
[vmax,vmin]	Matching	Iteraciones	Tiempo	Error del coste
[0,5;-0,5]	si	56	36,22s	0,082
[1;-1]	si	28	21,48s	0,081
[2;-2]	si	43	32,75s	0,084
Resumen de datos				
[vmax,vmin]	Matching	Iteraciones	Tiempo	Error del coste
[0,5;-0,5]	si	49	39,94s	0,203
[1;-1]	si	36	32,25s	0,204
[2;-2]	si	36,25	33,95s	0,208

Como se puede ver representado en la tabla velocidades máximas y mínimas por debajo de [1,-1] implica mayor número de iteraciones y mayor tiempo de ejecución. Por otro lado los tiempos de ejecución e iteraciones no varían demasiado entre [1,-1] y [2,-2]. Nos quedaremos con el límite en velocidades de [1,-1] ya que presenta una ligera ventaja.

5.3. Resultado scan matching con PSO

Como se ha visto apartado anterior, el problema del scan matching se ha podido resolver a través del algoritmo del PSO. Para su configuración final se han tomado valores de velocidades máxima y mínima de [1,-1] y tamaño de la nube de 50 partículas, ya que es lo que se ha demostrado que optimiza en mayor medida el algoritmo para este problema. Como se ve en los pares de escaneos comparados tenemos diferencias entre *frames* desde 1 (ejemplo: 500-501) hasta 10 (ejemplo: 1000-1010), lo que muestra en cierta medida la robustez del método. Por otro lado la eficacia del método siempre estará supeditada a la diferencia entre los *frames*, ya que cuanto más difieren tanto más difícil es resolver el problema de matching entre ellos.
Se van a exponer a continuación algunos experimentos y su resultado. Para ello se mostraran 2 ejemplos del proceso de scan matching, se muestran las imágenes RGB, las imágenes tras el filtrado de color para el preprocesamiento y el resultado de su matching final.

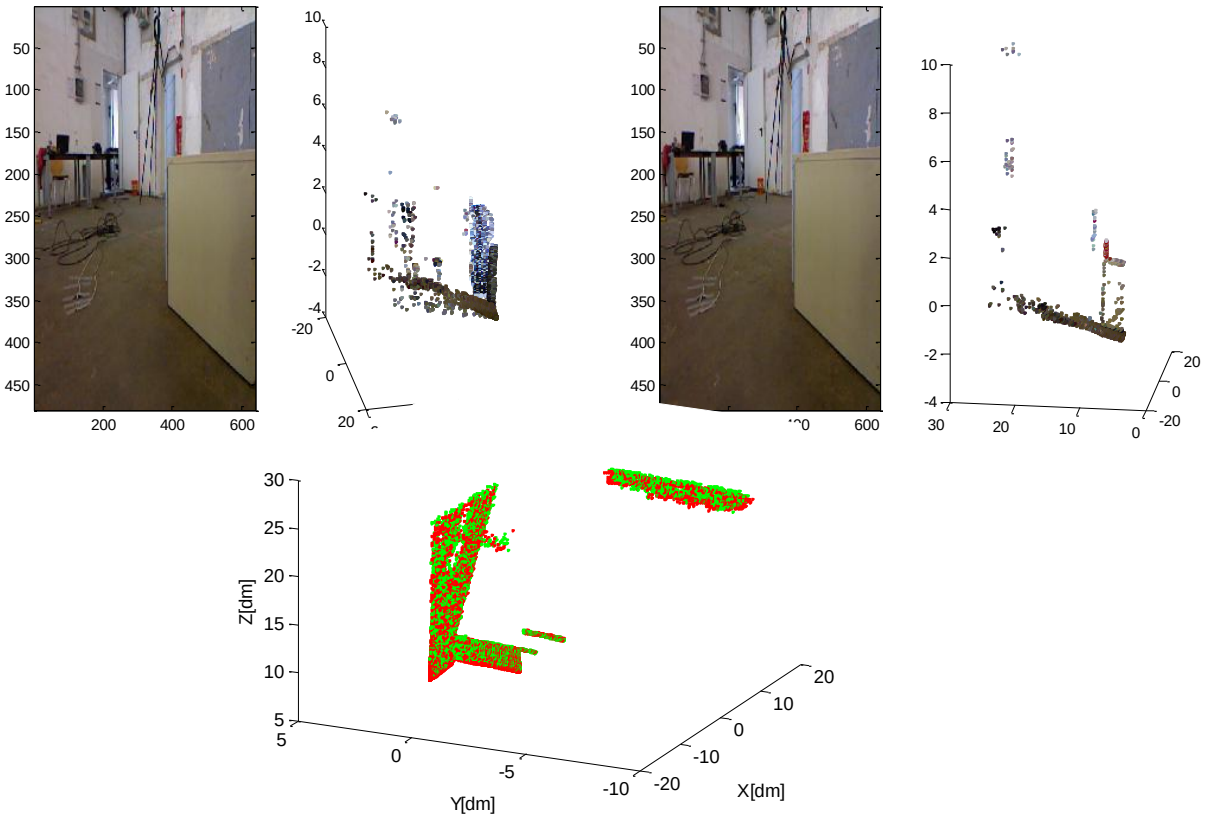

Figura 5.1: arriba a la izquierda: imagen 1000 RGB y filtrada; arriba a la derecha: imagen 1010 RGB y filtrada; abajo: resultado del matching de las 2 imágenes anteriores.

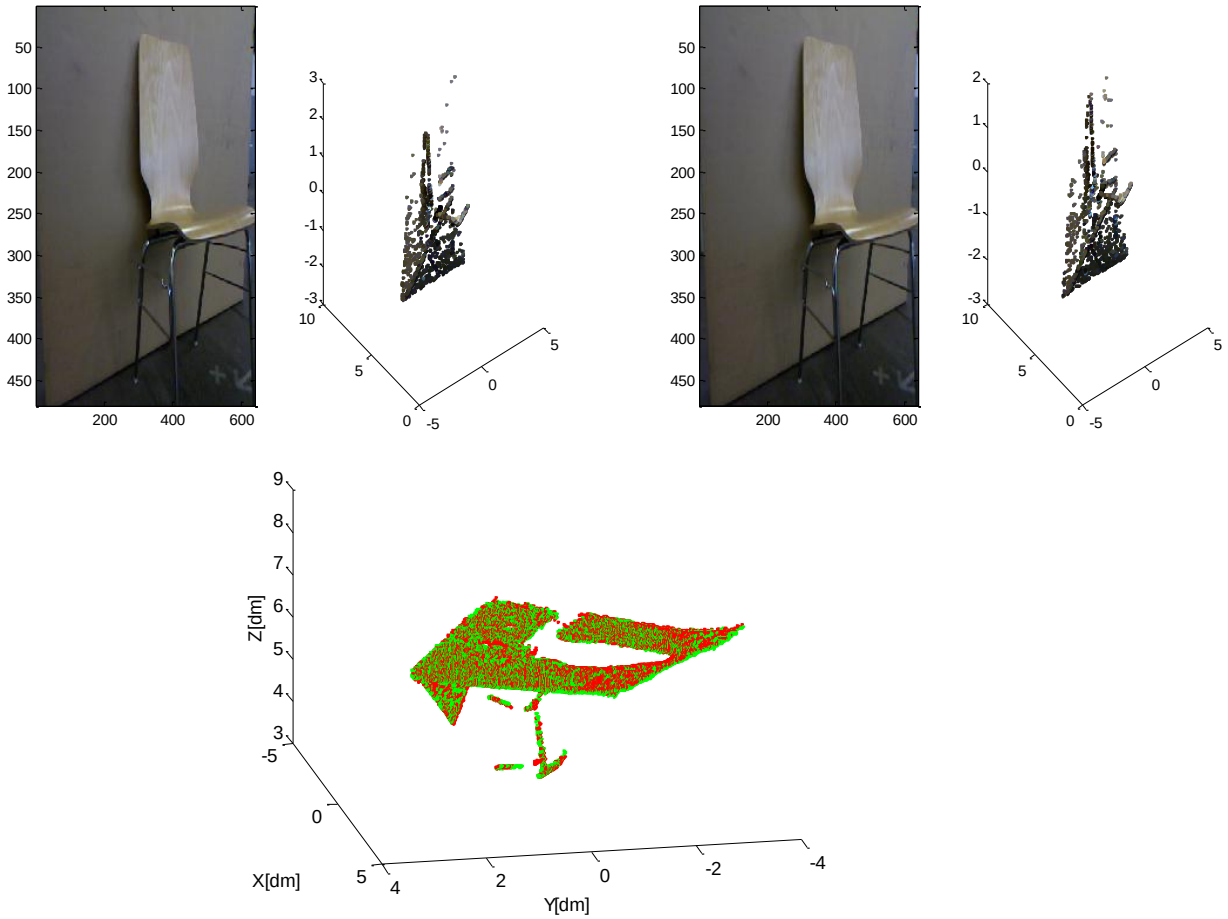

Figura 5.2: arriba izqda.: imagen 500 RGB y filtrada; arriba dcha.: imagen 501 RGB y filtrada; abajo: matching de las 2 imágenes.

Como se puede ver en las figuras de los ejemplos anteriores el matchig se consigue sin problemas.

5.4. Análisis del valle de convergencia

En este punto se va a mostrar un experimento interesante que se suele analizar en los problemas de scan matching, este es el análisis del valle de convergencia.

Este experimento pretende comprobar el resultado del scan matching cuando se le introducen diferentes *offset* de rotación y traslación. El interés de este experimento radica en ver lo robusto que es el algoritmo cuando se le aplican giros bruscos. Hay que remarcar que para este experimento se han aplicado condiciones muy restrictivas. Si en algunos experimentos como el de Magnuson *et al.* [32] los *offset* que se aplican son de 1m, en este caso se van a aplicar *offset* de 2m. además de ángulos de rotación de entre 80º y -80º. Los resultados del experimento se presentan en la figura 5.3.

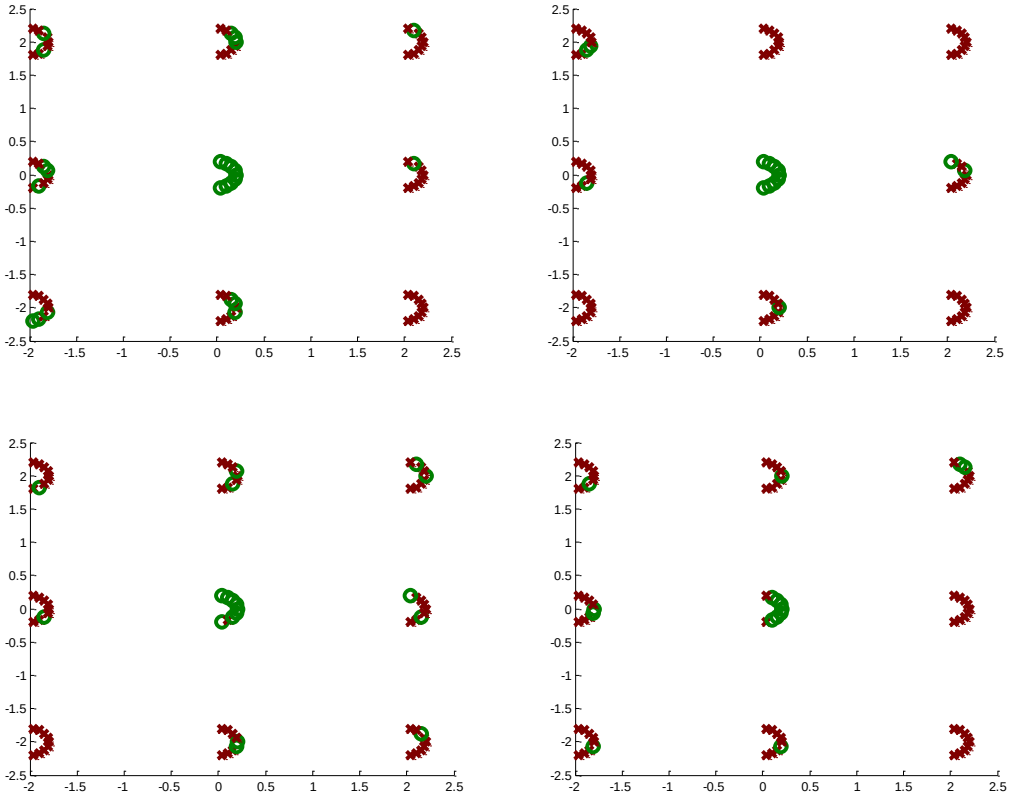

Figura 5.3: arriba izqda.: solución entre escaneos 1000-1010; arriba dcha.: solución entre escaneos 50-60; abajo izqda.: solución entre escaneos 1450-1451; abajo dcha.: solución entre escaneos 500-501

En esta figura cada semicírculo representa un *offset* de traslación y cada marca individual de cada semicírculo representa una rotación de ±20º para esa traslación. Los resultados satisfactorios se representan en verde con un círculo y los fallidos en rojo con una cruz.
El porcentaje de acierto en cada caso es el cociente entre los resultados favorables y el total de resultados multiplicado por 100.

El resultado de los porcentajes de acierto se puede ver en la tabla 5.3

Tabla 5.3: Resultados del análisis del valle de convergencia mediante PSO

Números de escaneo	Resultado
1000-1010	30,8%
50-60	18,51%
1450-1451	24,7%
500-501	18,5%

- **Comparación entre DE y PSO en el análisis del valle de convergencia**

En este apartado se va a exponer una comparativa entre el algoritmo de Differential Evolution (DE) [5] y del Particle Swarm Optimization como solución al scan matching respecto del análisis del valle de convergencia. En la tabla 5.4 se pueden ver los porcentajes de acierto del DE frente al PSO.

Tabla 5.4: Resultados de análisis del valle de convergencia del PSO frente al DE.

Números de escaneo	Resultado PSO	Resultado DE
1000-1010	30,8%	69,14%
50-60	18,51%	58,02%
1450-1451	24,7%	67,90%
500-501	18,5%	27,16%

Como se puede ver, los resultados obtenidos por Martín *et al.*[5] son más satisfactorios que los obtenidos mediante el PSO. Esto muestra que el método mediante el DE es un algoritmo más robusto frente a traslaciones y rotaciones bruscas. Ha de destacarse, por otro lado, el extenso trabajo realizado por Martín *et al*. en la creación de un algoritmo tan robusto.

Capítulo 6

Conclusiones y trabajo futuro

En el presente proyecto se ha propuesto una nueva forma de resolver el problema de scan matching, en particular mediante el algoritmo de Particle Swarm Optimization (PSO). Se han utilizado además las propiedades del color (diferencias de ΔE) en la fase de preprocesamiento de imágenes para optimizar el scan matching.
El método se ha probado condiciones reales, analizando la correlación entre imágenes de una serie de datos tomadas por la Kinect.
Se han obtenido resultados favorables demostrando que el PSO es un algoritmo eficiente en la resolución del problema de scan matching. Además se han estudiado parámetros como el número de partículas de la nube y la máxima y mínima velocidad alcanzable por dichas partículas que optimizan el algoritmo en cuanto a fiabilidad y coste computacional.
Ha de resaltarse también la inferior proporción de correlaciones favorables entre escaneos, bajo el análisis del valle de convergencia, que mediante el DE desarrollado por Martín *et al.* [5], lo que da pie a posibles investigaciones futuras en cuanto a su optimización.

Tras comprobar que el algoritmo del PSO puede ser aplicado para resolver el problema del scan matching, se abre un nuevo campo de posibles futuros análisis, algunos de ellos podrían ser:

- En lo que respecta al PSO aplicado al problema de scan matching un experimento interesante sería comparar el resultado del matching según distintas posibles topologías de población de las partículas en la nube (no tratadas en el presente proyecto).

- Encontrar una óptima parametrización, para lograr que el análisis del valle de convergencia resultase en mayor proporción de correlaciones favorables entre escaneos, lo que daría un método más robusto.
- Comparar los resultados obtenidos con un preprocesamiento de ΔE de puntos característicos frente a la toma de la nube entera de puntos sin preprocesamiento.
- Incluir las propiedades del color no solo en la parte de preprocesamiento de imágenes sino como elemento clave en la correlación de puntos según diferencias en el color para el caso de scan matching con PSO.

Bibliografía

[1] Business insider. *The robotics market report: The fast-multiplying opportunities in consumer, industrial, and office robots.*<http://www.businessinsider.com/growth-statistics-for-robots-market-2015-2>[Consulta: mayo de 2015]

[2] Rich, E., Night, K., (1991). *Artificial intelligence.*McGraw-Hill

[3] Leonard, J.J y Durrant-Whyte, H. (1991) "Mobile Robot Localization by Tracking Geometric Beacons". *IEEE Transaction on Robotics and Automation*, vol. 7, p. 376-382.

[4] Smith, R., Self, M., Cheesman, P., 1986. "Estimating Uncertain Spatial Relationships in Robotics". *Proceedings of the Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI' 86),* (New York, NY), p. 267-288.

[5] Martín, F., Miró, J. V., Moreno, L. (2014). "Towards exploiting the advantages of colour in scan matching". En *ROBOT2013: First Iberian Robotics Conference* (p. 217-231). Springer International Publishing.

[6] Thrun, S. (2002). "Robotic mapping: A survey". *Exploring artificial intelligence in the new millennium*, p. 1-35.

[7] Lu, F., Milios, E. (1997). "Robot pose estimation in unknown environments by matching 2d range scans". *Journal of Intelligent and Robotic Systems*, *18*(3), p. 249-275.

[8] Tomono, M. (2004). "A scan matching method using euclidean invariant signature for global localization and map building". En *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on* (Vol. 1, p. 866-871). IEEE.

[9] Aghamohammadi, A. A., Taghirad, H. D., Tamjidi, A. H., Mihankhah, E. (2007). "Feature-Based Laser Scan Matching For Accurate and High Speed Mobile Robot Localization". En *EMCR*.

[10] Nüchter, A., Lingemann, K., Hertzberg, J., Surmann, H. (2007). "6D SLAM—3D mapping outdoor environments". *Journal of Field Robotics*, *24*(8-9), p. 699-722.

[11] Zhang, Z. (1994). "Iterative point matching for registration of free-form curves and surfaces". *International journal of computer vision*, *13*(2), p. 119-152.

[12] Besl, P. J., McKay, N. D. (1992). "Method for registration of 3-D shapes". En *Robotics-DL tentative* (p. 586-606). International Society for Optics and Photonics.

[13] Rusinkiewicz, S., Levoy, M. (2001). "Efficient variants of the ICP algorithm". En *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on* (p. 145-152). IEEE.

[14] Biber, P., Straßer, W. (2003). "The normal distributions transform: A new approach to laser scan matching". En *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*(Vol. 3, p. 2743-2748). IEEE.

[15] Magnusson, M., Lilienthal, A., Duckett, T. (2007). "Scan registration for autonomous mining vehicles using 3D-NDT". *Journal of Field Robotics*, *24*(10), p. 803-827.

[16] Eberhart, R. C., Kennedy, J. (1995). "A new optimizer using particle swarm theory". En *Proceedings of the sixth international symposium on micro machine and human science* (Vol. 1, p. 39-43).

[17] Heppner, H., Grenander, U. (1990). "A stochastic non-linear model for coordinated bird flocks". En S.Krasner (Ed.), *The ubiquity of chaos* (p. 233-238). Washington: AAAS.

[18] Reynolds, C. W. (1987). "Flocks, herds and schools: A distributed behavioral model". En *ACM Siggraph Computer Graphics* (Vol. 21, No. 4, p. 25-34). ACM.

[19] Engelbrecht, A.P., (2005). *Fundamentals of computational swarm intelligence.* Sudáfrica: Wiley.

[20] Van Den Bergh, F. (2006). *An analysis of particle swarm optimizers* (Doctoral dissertation, University of Pretoria).

[21] Eberhart, R.C., Simpson, P.K., Dobbins, R.W. (1996). *Computational intelligence PC tools.* San Diego: Academic press professional.

[22] Schutte, J. F., Groenwold, A. A. (2003). Sizing design of truss structures using particle swarms. *Structural and Multidisciplinary Optimization*, *25*(4), p.261-269.

[23] Fan, H. (2002). A modification to particle swarm optimization algorithm. *Engineering Computations*, *19*(8), 970-989.

[24] Shi, Y., Eberhart, R. (1998). "A modified particle swarm optimizer". En *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference* (p. 69-73). IEEE.

[25] Naka, S., Genji, T., Yura, T., Fukuyama, Y. (2001). "Practical distribution state estimation using hybrid particle swarm optimization". En *Power Engineering Society Winter Meeting, 2001. IEEE* (Vol. 2, p. 815-820). IEEE.

[26] Venter, G., Sobieszczanski-Sobieski, J. (2003). "Particle swarm optimization". *AIAA*

journal, *41*(8), p. 1583-1589.

[27] Clerc, M. (2001). Think locally, act locally: The way of life of cheap-PSO, an adaptative PSO. <http://clerc.maurice.free.fr/pso/ > [Consulta: diciembre 2014]

[28] Clerc, M. (1999). "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization". En *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* (Vol. 3). IEEE.

[29] Eberhart, R. C., Shi, Y. (2000). "Comparing inertia weights and constriction factors in particle swarm optimization". En *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on* (Vol. 1, p. 84-88). IEEE.

[30] Van den Bergh, F., Engelbrecht, A. P. (2001). "Effects of swarm size on cooperative particle swarm optimizers". En *Proceedings of the genetic and evolutionary computation conference*, p. 892-899.

[31] Poli, R., Kennedy, J., Blackwell, T. (2007). "Particle swarm optimization". *Swarm intelligence*, *1*(1), p.33-57.

[32] Magnuson, M., Andreasson, H., Nüchter, A., Lilienthal, A.J. (2009). "Apperance-bassed loop detection from 3D laser data using the normal distributions transform". En *Proceedings of the IEEE international conference on robotics and automation (ICRA'09)*